Mobile Service Architecture 2 Specification

Public Review Version 0.19 – 09-November-2008

Java Community ProcessSM

Java™ Platform, Micro Edition

Keilalahdentie 2-4
P.O. Box 226
FIN-00045 Nokia Group
Finland

Vodafone House
The Connection
Newbury, Berkshire, RG14 2FN
United Kingdom

JSR 249 Expert Group

jsr-249-comments@jcp.org

RESEARCH AND EVALUATION LICENSE

The JSR-249 Specification is lead by Nokia and Vodafone Group Services Limited ("Vodafone"). Nokia and Vodafone have agreed that Nokia is entitled to act as the sole licensor for the Specification and grant the licenses on the terms of this License.

NOKIA CORPORATION ("NOKIA") IS WILLING TO LICENSE THIS SPECIFICATION TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT ("LICENSE"). PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE CAREFULLY. BY ACCESSING OR USING THE SPECIFICATION YOU WILL BE BOUND BY THE TERMS OF THIS LICENSE.

In this License, "Specification Lead" shall mean Nokia and "You" shall mean the individual downloading the Specification and accepting this License and where such individual downloads the Specification for other than private use, the legal entity represented by such individual.

JSR 249 - Mobile Service Architecture 2 ("Specification")

Version:	0.19
Status:	**Public Review**
Specification Lead:	Nokia Corporation, Vodafone Group Services Limited
Release:	November 9th, 2008

1. NOTICE; LIMITED LICENSE GRANTS

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

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

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

2. TRADEMARKS

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

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

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

3. DISCLAIMER OF WARRANTIES

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

3.2 THE SPECIFICATION MAY INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SPECIFICATION LEAD MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of

such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

4. LIMITATION OF LIABILITY

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

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

5. EXPORT CONTROL

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

6. RESTRICTED RIGHTS LEGEND

6.1 Note to U.S. Government Users. The Specification is a “Commercial Items”, as that term is defined at 48 C.F.R. 2. 101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation”, as such terms are used in 48 C.F.R. 12.212 or 48 C.F.R. 227.7202, as applicable. Consistent with 48 C.F.R. 12.212 or 48 C.F.R. 227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software Documentation are being licensed to U.S. Government end users a) only as Commercial Items and b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.

REQUEST FOR FEEDBACK

The JSR 249 Expert Group (EG) welcomes all feedback on the contents of this Public Review version of the Mobile Service Architecture 2 Specification.

This Public Review feedback should be provided to the EG e-mail address:

- **jsr-249-comments@jcp.org**

Please see specification section '3.7 Feedback' for details on how the EG will handle feedback and contributions from the review.

The EG requests reviewers to provide feedback especially on the following specification areas:

1. **Which JSRs in the MSA 2 Standard Platform (SP) are important/less important**
    - **Some EG members have expressed their concern on the memory footprint of the MSA 2 SP implementation. This could lead to MSA 2 SP not being deployed as widely as MSA 2 Entry Platform (EP) in mobile devices.**
    - **EG is considering to move one or several JSRs from the MSA 2 SP into the MSA 2 Advanced Platform (AP).**
    - **Reviewers should provide feedback on which JSRs in the currently defined MSA 2 SP are most important and which are least important. This feedback will be used when the EG reconsiders the set of JSRs included in MSA 2 SP during and after the Public Review.**
2. **Media requirements defined in sections:**
    - **6.3.3.13 Media Format Support**
    - **6.3.3.14 MIME types and Controls for Media Playback**

    The EG would like to receive feedback on the media formats envisioned to be supported in MSA 2 EP, SP and AP compliant devices. Which formats are more important than others? Is something missing? What could be removed?

Note: As the specification work in the EG is still ongoing, the JSR 249 EG plans to publish at least one update version of this specification during the Public Review.

A discussion board for Mobile Service Architecture has also bee established at:

- **http://discussion.forum.nokia.com/forum/forumdisplay.php?f=185**

This discussion board can also be used to discuss the contents of the specification, but all official feedback to the JSR 249 EG should be provided to the Expert Group e-mail address (jsr-249-comments@jcp.org).

Contents:

1. SCOPE 8
2. REFERENCES 9
2.1 NORMATIVE REFERENCES 9
3. TERMINOLOGY AND CONVENTIONS 12
3.1 CONVENTIONS 12
3.2 DEFINITIONS 12
3.3 ABBREVIATIONS 12
3.4 TYPOGRAPHIC CONVENTIONS 16
3.5 APPROVED VERSION HISTORY 16
3.6 CURRENT VERSION HISTORY 16
3.7 FEEDBACK 17
4. INTRODUCTION (INFORMATIVE) 19
4.1 DESIGN GOALS 19
4.2 SPECIFICATION STRUCTURE 21
4.3 EXPERT GROUP 21
5. MSA 2 COMPONENT JSRS (NORMATIVE) 22
6. ADDITIONAL CLARIFICATIONS (NORMATIVE) 27
6.1 PDA OPTIONAL PACKAGES FOR THE J2ME PLATFORM (JSR 75) 27
6.2 JAVA APIS FOR BLUETOOTH (JSR 82) 43
6.3 MOBILE MEDIA API (JSR 135) 46
6.4 CONNECTED LIMITED DEVICE CONFIGURATION (JSR 139) AND CONNECTED DEVICE CONFIGURATION (JSR 218) 58
6.5 J2ME WEB SERVICES (JSR 172) 62
6.6 SECURITY AND TRUST SERVICES API (JSR 177) 64
6.7 SIP API FOR J2ME (JSR 180) 68
6.8 WIRELESS MESSAGING API (JSR 205) 69
6.9 CONTENT HANDLER API (JSR 211) 76
6.10 PAYMENT API (JSR 229) 82
6.11 ADVANCED MULTIMEDIA SUPPLEMENTS (JSR 234) 83
6.12 MOBILE INTERNATIONALIZATION API (JSR 238) 84
6.13 JAVA BINDING FOR THE OPENGL ES API (JSR 239) 85
6.14 MOBILE SENSOR API (JSR 256) 86
6.15 CONTACTLESS COMMUNICATION API (JSR 257) 89
6.16 MOBILE USER INTERFACE CUSTOMIZATION API (JSR 258) 90
6.17 MIDP 2.1 (JSR 118) AND MIDP 3 (JSR 271) 91
6.18 MOBILE BROADCAST SERVICE API (JSR 272) 102
6.19 XML API FOR JAVA ME (JSR 280) 103
6.20 IMS SERVICES API (JSR 281) 104
6.21 SCALABLE 2D VECTOR GRAPHICS API 2.0 FOR JAVA ME (JSR 287) 105
6.22 JAVA LANGUAGE & XML UI MARKUP INTEGRATION (JSR 290) 106
6.23 LOCATION API 2.0 (JSR 293) 107
6.24 MOBILE 3D GRAPHICS API 2.0 (JSR 297) 108
6.25 REQUIREMENTS INHERITED FROM JTWI 1.0 (JSR 185) 110
7. ADDITIONAL REQUIREMENTS (NORMATIVE) 122
7.1 HARDWARE REQUIREMENTS 122

7.2 SECURITY REQUIREMENTS 124
8. RECOMMENDATIONS AND GUIDELINES (INFORMATIVE) 139
8.1 GUIDELINE FOR APPLICATIONS REFERRING NON-MANDATORY APIS 139
9. ROADMAP (INFORMATIVE) 142
APPENDIX A. SUMMARY TABLES (INFORMATIVE) 143
A.1 SYSTEM PROPERTIES 143
A.2 NETWORK PROTOCOLS AND CONTENT FORMATS 147
A.3 HARDWARE REQUIREMENTS AND RECOMMENDATIONS 150
A.4 SECURITY PERMISSIONS AND FUNCTION GROUPS 152
APPENDIX B. LIST OF JAVA ME JSRS (INFORMATIVE) 157

1. Scope

This Mobile Service Architecture 2 (MSA 2) Specification (JSR 249) defines the next version of the Mobile Service Architecture (MSA) defined in JSR 248. The specification defines the next step in the Java platform evolution for mobile handsets based on both the Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC) of the Java™ Platform, Micro Edition (Java ME). MSA 2 Specification defines the mandatory sets of JSRs in device implementations together with a number of optional JSRs for specific technology and application areas. In addition the MSA 2 specification also defines a more limited set of JSRs for low end devices previously unaddressed by MSA.

This specification defines a normative collection of component JSRs that together with Additional Clarifications and Additional Requirements define version 2 of the the Mobile Service Architecture. Additionally, this document includes a set of recommendations to help the reader create an optimal MSA 2 compliant implementation.

2. References

2.1 Normative References

[AMR] 3GPP TS 26.071 “Adaptive Multi-Rate (AMR) Speech Codec; General Description”

[JAR] JAR File Specification for JDK 1.3.1, http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html

[JPEG] “Information Techology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines”, ISO/IEC 10918-1, 1994

[JSR75] “PDA Optional Packages for the J2ME Platform”, Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=75

[JSR82] “Java APIs for Bluetooth”, Version 1.1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=82

[JSR118] “Mobile Information Device Profile”, Version 2.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=118

[JSR135] “Mobile Media API”, Version 1.2, Java Community Process, http://www.jcp.org/en/jsr/detail?id=135

[JSR139-1.1] “Connected Limited Device Configuration”, Version 1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=139

[JSR139-1.1.1] “Connected Limited Device Configuration”, Version 1.1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=139

[JSR172] “J2ME Web Services Specification”, Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=172

[JSR177] “Security and Trust Services API for J2ME”, Version 1.0.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=177

[JSR179] “Location API for J2ME™”, Version 1.0.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=179

[JSR180] “SIP API for J2ME”, Version 1.1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=180

[JSR185] “Java™ Technology for the Wireless Industry”, Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=185

[JSR205] “Wireless Messaging API 2.0”, Version 2.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=205

[JSR211] “Content Handler API”, Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=211

[JSR218] “Connected Device Configuration (CDC)”, Version 1.1.2, Java Community Process, http://jcp.org/en/jsr/detail?id=218

[JSR226] “Scalable 2D Vector Graphics API for J2ME”, Version 1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=226

[JSR234] "Advanced Multimedia Supplements", Version 1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=234

[JSR238] "Mobile Internationalization API", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=238

[JSR239] "Java Binding for the OpenGL ES API", Version 1.0.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=239

[JSR256] "Mobile Sensor API", Version 1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=256

[JSR257] "Contactless Communication API", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=257

[JSR258] "Mobile User Interface Customization API", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=258

[JSR271] "Mobile Information Device Profile 3", Version 3.0, Java Community Process, http://jcp.org/en/jsr/detail?id=271

[JSR272] "Mobile Broadcast Service API for Handheld Terminals", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=272

[JSR279] "Service Connection API for Java ME", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=279

[JSR280] "XML API for Java ME", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=280

[JSR281] "IMS Services API", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=281

[JSR287] "Scalable 2D Vector Graphics API 2.0 for Java ME", Java Community Process, http://jcp.org/en/jsr/detail?id=287

[JSR290] "Java Language & XML User Interface Markup Integration", Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=290

[JSR293] "Location API 2.0", Version 2.0, Java Community Process, http://jcp.org/en/jsr/detail?id=293

[JSR297] "Mobile 3D Graphics API 2.0", Version 2.0, Java Community Process, http://jcp.org/en/jsr/detail?id=297

[PNG] PNG (Portable Network Graphics) Specification, Version 1.0, W3C Recommendation, October 1, 1999,, http:// www.w3.org/TR/REC-png.html. Also available as RFC 2083, http://www.ietf.org/rfc/rfc2083.txt

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996, http://www.ietf.org/rfc/rfc2045.txt

[RFC2046] "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed, N. Borenstein, November 1996, http://www.ietf.org/rfc/rfc2046.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2326] "Real Time Streaming Protocol (RTSP)", H. Schulzrinne, A. Rao, R. Lanphier, April 1998, http://www.ietf.org/rfc/rfc2326.txt

[RFC2368] "The mailto URL scheme", P. Hoffman, L. Masinter, J. Zawinski, July 1998, http://www.ietf.org/rfc/rfc2368.txt

[RFC2616] "Hypertext Transfer Protocol – HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, The Internet Society, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC3966] "The tel URI for Telephone Numbers", H. Schulzrinne, December 2004, http://www.ietf.org/rfc/rfc3966.txt

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, January 2005, http://www.ietf.org/rfc/rfc3986.txt

[SP-MIDI_1] Scalable polyphony MIDI specification, version 1.0, RP-034. The MIDI Manufacturers Association, Los Angeles, CA, USA, 2002, http://www.midi.org/about-midi/abtspmidi.shtml

[SP-MIDI_2] Scalable polyphony MIDI device 5–24 note profile for 3GPP, version 1.0, RP-035. The MIDI Manufacturers Association, Los Angeles, CA, USA, 2002, http://www.midi.org/about-midi/abtspmidi.shtml

3. Terminology and Conventions

3.1 Conventions

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

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

Additional Clarifications	Requirements or recommendations that complement component JSR specifications.
Compliant	Conforms to the requirements of this specification.
Compliant device	A device that has an implementation that conforms to the requirements of this specification.
Compliant implementation	Implementation that conforms to the requirements of this specification.
Component JSR	A Java specification (JSR) included in one of the MSA 2 platforms.
Conditionally mandatory	MUST be implemented according to the specification if the given condition is fulfilled.
Mandatory	MUST be implemented according to the specification.
MSA 2 compliant implementation	Implementation that conforms to the requirements of this specification.
MSA 2 compliant device	A device that has an implementation that conforms to the requirements of this specification.
MSA 2 TCK	The documentation and software to be used for testing whether an implementation is compliant with this specification. It includes: 1) Test cases that can be used to guarantee conformance with the requirements specified in this specification 2) Instructions and guidance as to how to run these test cases The individual component JSR TCKs SHALL NOT be included in the MSA 2 TCK delivery package.

3.3 Abbreviations

3GPP	3rd Generation Partnership Project
3GP	multimedia container format defined by 3GPP
3GPP	Third Generation Partnership Project

AAC	Advanced Audio Coding
AMR	Adaptive Multi Rate (audio coding)
AMR-NB	AMR Narrow Band
AMR-WB	AMB Wide Band
AMS	Application Management System
AP	Advanced Platform (in MSA 2)
APDU	Application Protocol Data Unit
API	Application Programming Interface
BCC	Blind Carbon Copy
CBS	Cell Broadcast Service
CDC	Connected Device Configuration
CHAPI	Content Handler API
CLDC	Connected Limited Device Configuration
DOM	Document Object Model
EG	Expert Group
EP	Entry Platform (in MSA 2)
FC	FileConnection
FM	Frequency Modulation
GMT	Greenwich Mean Time
GSM	Global System for Mobile communications
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol over Secure Socket Layer
IANA	Internet Assigned Numbers Authority
IMS	IP Multimedia Subsystem
J2ME	Java 2 Platform, Micro Edition
JAD	Java Application Descriptor
JAR	Java Archive
Java ME	Java Platform, Micro Edition
JAXP	Java API for XML Processing
JCP	Java Community Process
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Experts Group
JSR	Java Specification Request
JTWI	Java Technology for the Wireless Industry

L2CAP	Logical Link Control and Adaptation Protocol
M3G	Mobile 3D Graphics
MIDI	Musical Instrument Digital Interface
MIDlet	MIDP Application
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MMAPI	Mobile Media API
MMS	Multimedia Messaging Service
MMSC	Multimedia Messaging Service Center
MP3	MPEG-1 Audio Layer 3
MP4	MPEG-4 Part 14 (multimedia container format)
MSA	Mobile Service Architecture
NFC	Near Field Communication
OBEX	Object Exchange
OpenGL ES	Open Graphics Library for Embedded Systems
OTA	Over The Air
PCM	Pulse-Code Modulation
PIM	Personal Information Management
PKI	Public Key Infrastructure
PNG	Portable Network Graphics
RBDS	Radio Broadcast Data System
RDS	Radio Data System
RFC	Request For Comments
RFID	Radio-Frequency Identification
RI	Reference Implementation
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RMS	Record Management System
RTSP	Real Time Streaming Protocol
SATSA	Security And Trust Services API
SDP	Session Description Protocol
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Message Service

SMSC	Short Message Service Center
SP	Standard Platform (in MSA 2)
SP-MIDI	Scalable Polyphony MIDI
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
TCK	Technology Compatibility Kit
TCP	Transmission Control Protocol
TLS	Transport Layer Security
uDOM	Micro DOM
UDP	User Datagram Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USIM	Universal Subscriber Identity Module
UTF	Unicode Transformation Format
VHF	Very High Frequency
WAP	Wireless Application Protocol
WAV	Waveform Audio Format
WMA	Wireless Messaging API
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

3.4 Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123`	The names of package, class names, and other Java constructs	`java.lang.String` `java.util.HashTable.get()`

3.5 Approved Version History

Reference	Date	Description
JSR 248 version 1.00	27 September 2006	Mobile Service Architecture Specification Final Release
JSR 248 version 1.1.0b	18 August 2008	Mobile Service Architecture Specification Maintenance Release / Final Release 2

3.6 Current Version History

Document Identifier	Date	Description
Early Draft: JSR 248, version 0.50	08 April 2005	Initial version
Public Review: JSR 248 version 0.80	06 September 2005	Added new component JSRs, additional clarifications, security sections and hardware requirements/recommendations.
Proposed Final Draft: JSR 248 version 0.95	28 April 2006	Included MIDP 2.1, improved clarifications, added MSA Subset support, editorial changes.
Final Approval Ballot: JSR 248 version 1.00	27 September 2006	Final bug fixes and editorial changes.
Final Release: JSR 248 version 1.00	27 September 2006	Mobile Service Architecture Specification Final Release
Maintenance Release: JSR 248 version 1.1.0b	18 August 2008	Mobile Service Architecture Specification Maintenance Release / Final Release 2
JSR 249 EG Private Draft 0.01	31 December 2007	Initial EG internal draft for the Mobile Service Architecture 2 Specification summarizing the work status from year 2007
JSR 249 EG Private Draft 0.02	01 February 2008	EG private draft after incorporating minor corrections based on comments from the Expert Group.
JSR 249 EG Private Draft 0.03	13 February 2008	Minor corrections.
JSR 249 EG Private Draft 0.04	24 March 2008	1st EDR draft candidate for the EG.
JSR 249 EG Private Draft 0.05	26 March 2008	2nd EDR draft candidate for the EG.
JSR 249 Early Draft Review, Version 0.10	30 March 2008	Version published for Early Draft Review.

Document Identifier	Date	Description
JSR 249 Pre Public Review, Version 0.11	18 June 2008	EG private draft with changes originating from Early Draft Review comments and work in the EG.
JSR 249 Pre Public Review, Version 0.12	23 June 2008	EG private draft with a number of new clarifications added.
JSR 249 Pre Public Review, Version 0.13	4 July 2008	EG confidential draft with some more clarifications added and editorial changes made.
JSR 249 Pre Public Review, Version 0.14	27 August 2008	EG confidential draft with some improvements to clarifications and editorial changes.
JSR 249 Pre Public Review, Version 0.15	8 October 2008	EG confidential draft with some changes agreed by the EG.
JSR 249 Pre Public Review, Version 0.16	21 October 2008	EG confidential draft with editorial corrections, updated diagrams and HW requirements and an updated MIDP 3 compatible security section.
JSR 249 Pre Public Review, Version 0.17	31 October 2008	EG confidential draft with editorial corrections and new MMAPI media format requirements.
JSR 249 Pre Public Review, Version 0.18	5 November 2008	Candidate for Public Review with changes and additions agreed in the November 2008 MSA 2 face-to-face meeting.
JSR 249 Public Review, Version 0.19	9 November 2008	Public Review version.

3.7 Feedback

We are interested in improving this specification and welcome your comments and suggestions. You can email your comments to:

jsr-249-comments@jcp.org

Note: The following description summarizes the process that the MSA 2 Specification Leads and the MSA 2 Expert Group use to handle comments and suggestions:

MSA 2 accepts *comments* and *clarification proposals* (Contributions) through two main channels: 1) directly from the MSA 2 Expert Group members, and 2) from the MSA 2 specification feedback alias (see the e-mail address above). In each case, the Contribution is recorded and added as an Open Issue into a database that contains the Contribution, the Contributor (name of a company or a person who submitted the Contribution, or both), the number of the component JSR or the MSA 2 Specification section that it addresses, and other relevant information. These Open Issues are then discussed by the MSA 2 Expert Group. As a result of the discussions, possible new clarifications might be generated and added to the MSA 2 Specification.

MSA 2 Specification Leads make all clarifications related to a specific component JSR available to the Specification Leads of the respective component JSR so that the JSR Specification Leads can evaluate and provide feedback on the clarifications.

The component JSR Specification Lead has the right to produce a new Maintenance Release of the component JSR, and to include the proposed MSA 2 clarification or clarifications in the component JSR specification itself. If this happens within a schedule that is reasonable from the viewpoint of the MSA 2 release schedule, MSA 2 references the Maintenance Release directly, and does not include the clarification or

clarifications in the MSA 2 Specification itself. If the JSR Specification Lead is not able to produce a new Maintenance Release within a reasonable time, the component JSR Specification Lead can at any time later make a Maintenance Release, or a new JSR (for a major version), that will include the clarification or clarifications. When that happens, the component JSR TCK must be augmented to contain the necessary test or tests for the new clarification or clarifications. The next MSA 2 Specification release then references the Maintenance Release of the component JSR, and removes the clarification from the MSA 2 Specification and removes the corresponding test or tests from the MSA 2 TCK.

Important: All contributions sent to the MSA 2 Expert Group (comments, clarification proposals, and so on) are, by default, also contributions to the respective component JSR or JSRs. The MSA 2 Specification Leads also make those contributions available to the component JSR Specification Leads. If this is not desired by the contributor, this must be explicitly stated in the contribution.

4. Introduction (informative)

The Java ME community has developed a unified Java application environment standard for mobile phones as part of the JSR 248 Mobile Service Architecture specification. Mobile Service Architecture 2 Initiative continues the work by providing an updated platform specification needed by the mobile industry today.

MSA 2 Specification (JSR 249) addresses an even broader set of devices than MSA with more enhanced and diverse capabilities but continues its focus on high-volume mobile devices. This JSR broadens the architecture defined by MSA by adding support for new technologies and features that are already available or will become available in the foreseeable future. It also oversees compatibility with the JSR 248 MSA environment.

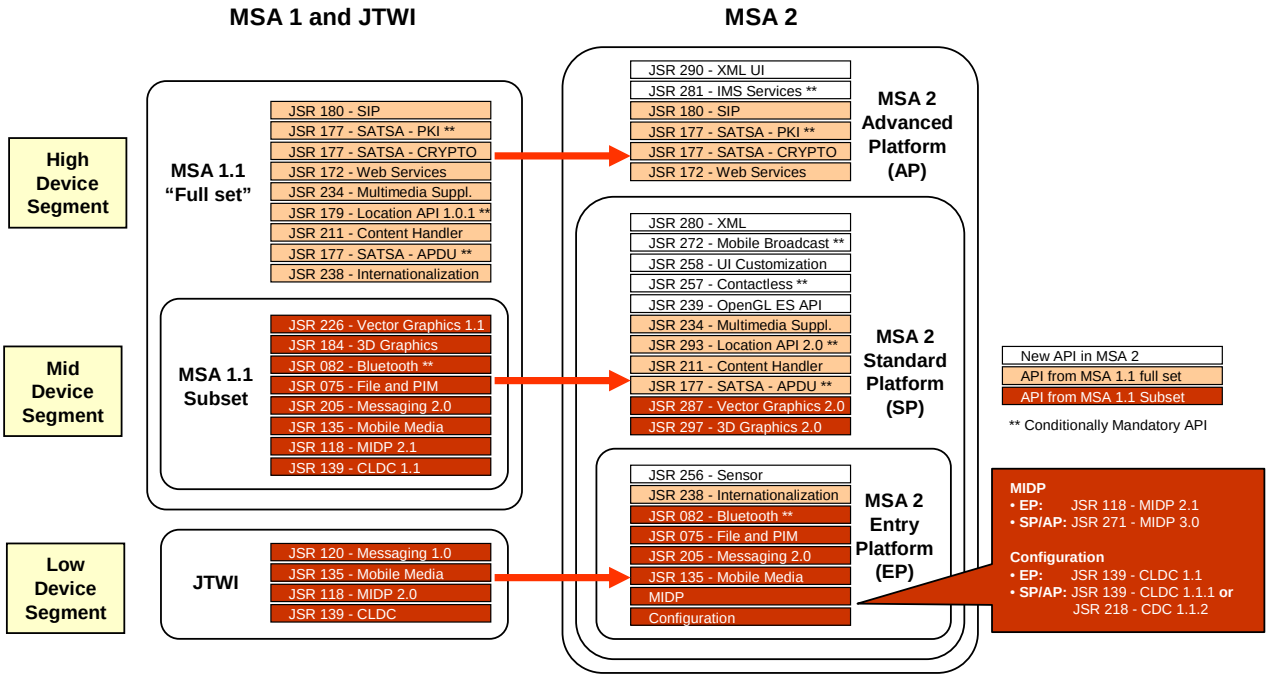
The MSA 2 Specification is a Java architecture definition that describes the essential Java client components of an end-to-end wireless environment. The MSA 2 Specification defines a set of Java ME technologies and shows how these technologies must be be correctly integrated in a mobile device to create an optimal mobile Java platform. As a normative specification, the MSA 2 Specification produces compatibility requirements that are reflected in the MSA 2 TCK. Service and content providers can use the MSA 2 Specification as a guideline for application development and can benefit from better application portability between different MSA 2 compliant implementations.

4.1 Design Goals

The primary design goal of the MSA 2 Specification is to minimize fragmentation of mobile Java environments by defining a predictable and highly interoperable application and service environment for developers. To achieve this goal, this specification defines Component JSRs, Additional Clarifications and Additional Requirements and Recommendations and Guidelines for implementers. The specification guides platform implementers in their efforts to maximize the interoperability of their implementation with other MSA 2 compliant implementations and application developers in their efforts to make sure that their applications will work in a wide variety of MSA 2 compliant devices.

The second design goal for MSA 2 is to enable its use in a wide variety of different markets and customer segments. This is achieved by allowing CDC in addition to CLDC as the underlying configuration, and well-defined conditionality for features that may not be available on all MSA 2 compliant devices.

The third design goal of MSA 2 is to provide a backward compatible evolution path from MSA 1 (JSR 248) and JTWI (JSR 185). This evolution path is illustrated in the following diagram.


MSA 1 and JTWI
MSA 2
High Device Segment
Mid Device Segment
Low Device Segment
MSA 1.1 "Full set"
JSR 180 - SIP
JSR 177 - SATSA - PKI **
JSR 177 - SATSA - CRYPTO
JSR 172 - Web Services
JSR 234 - Multimedia Suppl.
JSR 179 - Location API 1.0.1 **
JSR 211 - Content Handler
JSR 177 - SATSA - APDU **
JSR 238 - Internationalization
MSA 1.1 Subset
JSR 226 - Vector Graphics 1.1
JSR 184 - 3D Graphics
JSR 082 - Bluetooth **
JSR 075 - File and PIM
JSR 205 - Messaging 2.0
JSR 135 - Mobile Media
JSR 118 - MIDP 2.1
JSR 139 - CLDC 1.1
JTWI
JSR 120 - Messaging 1.0
JSR 135 - Mobile Media
JSR 118 - MIDP 2.0
JSR 139 - CLDC
JSR 290 - XML UI
JSR 281 - IMS Services **
JSR 180 - SIP
JSR 177 - SATSA - PKI **
JSR 177 - SATSA - CRYPTO
JSR 172 - Web Services
MSA 2 Advanced Platform (AP)
JSR 280 - XML
JSR 272 - Mobile Broadcast **
JSR 258 - UI Customization
JSR 257 - Contactless **
JSR 239 - OpenGL ES API
JSR 234 - Multimedia Suppl.
JSR 293 - Location API 2.0 **
JSR 211 - Content Handler
JSR 177 - SATSA - APDU **
JSR 287 - Vector Graphics 2.0
JSR 297 - 3D Graphics 2.0
MSA 2 Standard Platform (SP)
JSR 256 - Sensor
JSR 238 - Internationalization
JSR 082 - Bluetooth **
JSR 075 - File and PIM
JSR 205 - Messaging 2.0
JSR 135 - Mobile Media
MIDP
Configuration
MSA 2 Entry Platform (EP)
New API in MSA 2
API from MSA 1.1 full set
API from MSA 1.1 Subset
** Conditionally Mandatory API
MIDP
• EP: JSR 118 - MIDP 2.1
• SP/AP: JSR 271 - MIDP 3.0
Configuration
• EP: JSR 139 - CLDC 1.1
• SP/AP: JSR 139 - CLDC 1.1.1 or JSR 218 - CDC 1.1.2

4.2 Specification Structure

The MSA 2 Specification consists of the following main logical elements:

- **Component JSRs.** MSA 2 Specification describes the essential client components of an end-to-end wireless environment. It defines the sets of JSRs implemented by compliant implementations.
- **Additional Clarifications.** To improve predictability and interoperability, a description of each component JSR is accompanied by additional clarifications. Their purpose is to remove possible problems with the interpretation of component JSRs and minimize optionality whenever feasible.
- **Additional Requirements.** These are requirements related to hardware and security. Additional requirements provide implementers with more requirements and consequently improve backward compatibility, interoperability, and predictability of MSA 2 compliant implementations.
- **Recommendations and Guidelines.** These are suggestions for application developers and implementers of this specification on how to write applications for the MSA 2 environment and create an optimal MSA 2 compliant implementation.
- **Roadmap.** The MSA 2 roadmap aims to describe the future view of the mobile Java platform.

4.3 Expert Group

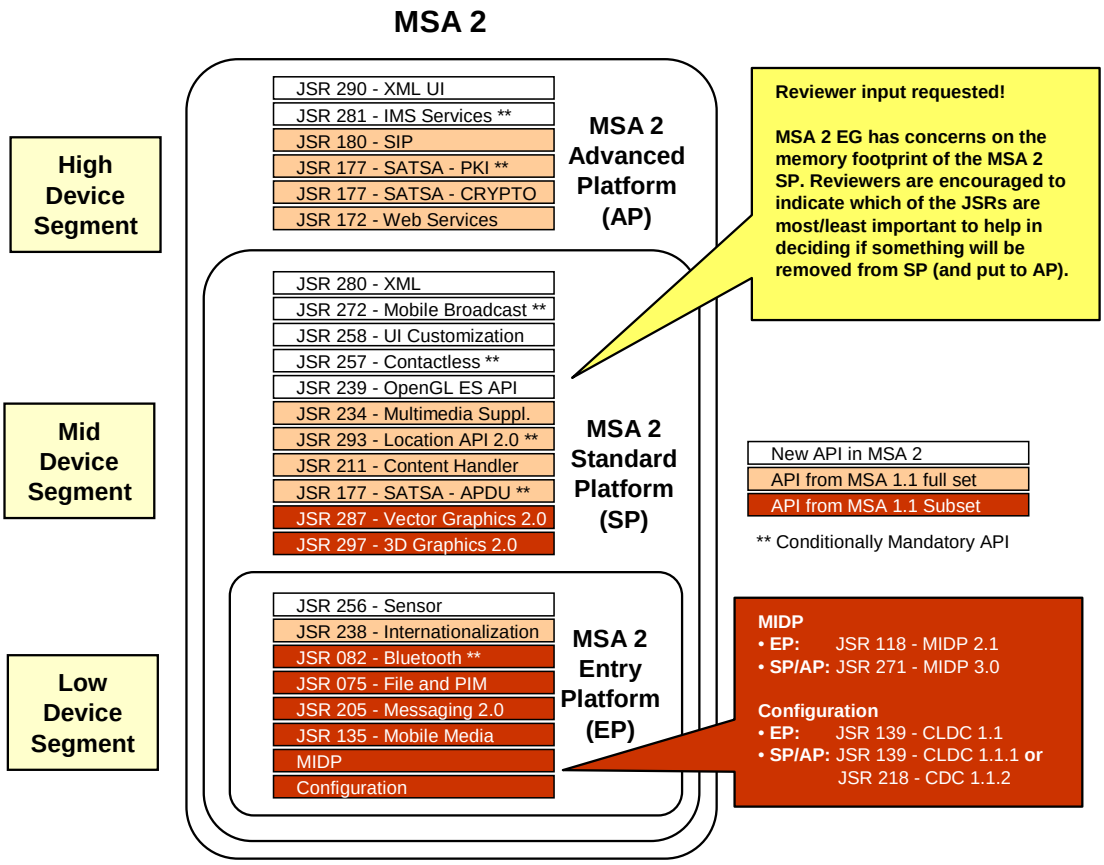
The MSA 2 Specification is a result of a focused effort conducted by an Expert Group representing the wireless industry across the world.

MSA 2 Expert Group members:

- Aplix Corporation
- BEA Systems
- BenQ
- China Mobile Communications Co. Ltd
- Cingular Wireless
- Ericsson AB
- Esmertec AG
- IBM
- Intel
- LG Electronics
- Motorola
- Nokia Corporation
- NTT DoCoMo
- Orange France SA
- Prosyst Software GmbH
- Research In Motion, LTD (RIM)
- Samsung Electronics Corporation
- Siemens AG
- Sony Ericsson Mobile Communications AB
- Sprint
- Sun Microsystems, Inc.
- T-Mobile International AG & Co. KG
- Telefonica
- TeliaSonera AB
- Vodafone Group Services Limited

5. MSA 2 Component JSRs (normative)

The following picture shows the components of the MSA 2 platform:



A compliant implementation MUST conform to the following requirements:

1. Implement every Component JSR of one of the API sets: MSA 2 Entry Platform (EP), MSA 2 Standard Platform (SP) or MSA 2 Advanced Platform (AP) as specified in the following table. This includes implementing every Conditionally Mandatory Component JSR if the corresponding condition is met. Any of the Component JSRs MAY be replaced by a later backward compatible version of the JSR.

2. Comply with all the Additional Clarifications for all implemented JSRs that are listed in the Additional Clarifications section. This includes clarifications for all JSRs in the MSA 2 Specification, not only the ones included in the MSA 2 platform (EP, SP or AP) that is being implemented.

3. Comply with all the Additional Requirements as specified in chapter “Additional Requirements”.

Some Component JSRs and their optional packages are *conditionally mandatory*. Additional Clarifications for these component JSRs include a statement describing the

condition on which an MSA 2 compliant implementation MUST implement the JSRs and comply with the clarifications outlined in the corresponding Additional Clarifications sections.

The following table specifies the Component JSRs and their optional packages included in different MSA 2 Platforms:

M = Mandatory, *C = Conditionally Mandatory*

Specification	Optional Packages or Features	Comments	MSA 2 EP	MSA 2 SP	MSA 2 AP
JSR 75 – PDA Optional Packages for the J2ME Platform Version 1.0 [JSR75]	**File API**	This API allows access to the file system available on a device.	M	M	M
	PIM API	This API provides access to Personal Information Management functionality.	M	M	M
JSR 82 – Java APIs for Bluetooth Version 1.1.1 [JSR82]	**Bluetooth API**	This API provides access to Bluetooth functionality.	C	C	C
	OBEX API	This API provides access to OBEX protocol, allowing the exchange of all kinds of objects such as files, pictures, calendar entries, and business cards.	C	C	C
JSR 118 – Mobile Information Device Profile Version 2.1 [JSR118]		This API is a fundamental component of MSA.	M		
JSR 135 – Mobile Media API Version 1.2 [JSR135]		This API provides a standard way to access media capabilities such as audio and video playback.	M	M	M
JSR 139 – Connected Limited Device Configuration* Version 1.1 [JSR139-1.1]		This API is a fundamental component of MSA.	M		
JSR 139 – Connected Limited Device Configuration Version 1.1.1 [JSR139-1.1.1]		This API is a fundamental component of MSA.		M	M

* A compliant implementation MAY use CDC instead of CLDC. For more information, see the MIDP 2.1 and MIDP 3.0 Specification and the definition of the Runtime-Execution-Environment JAD / JAR manifest attribute.

Specification	Optional Packages or Features	Comments	MSA 2 EP	MSA 2 SP	MSA 2 AP
JSR 172 – J2ME Web Services Specification Version 1.0 [JSR172]	**XML Parsing**	This package is provided by JSR 280.			
	Web Services (JAX RPC)	This API provides access to basic web services.			M
JSR 177 – Security and Trust Services API for J2ME Version 1.0.1 [JSR177]	**SATSA-CRYPTO**	This API provides a generic way to access cryptographic services.			M
	SATSA-APDU	This API provides a standard way to communicate with Smart Cards.		C	C
	SATSA-PKI	This API provides a generic way to access Public Key Infrastructure services.			C
	SATSA-JCRMI	Not part of MSA 2.			
JSR 180 – SIP API for J2ME Version 1.1.0 [JSR180]		This API provides a standard, low-level access to Session Initiation Protocol (SIP).			M
JSR 205 – Wireless Messaging API 2.0 Version 2.0 [JSR205]		This API provides access to messaging features such as SMS and MMS.	M	M	M
JSR 211 – Content Handler API Version 1.0 [JSR211]		This API enables the launching of Java applications based on content type.		M	M
JSR 218 – Connected Device Configuration Version 1.1.2 [JSR218]		Alternative for JSR 139 Connected Limited Device Configuration.			
JSR 234 – Advanced Multimedia Supplements Version 1.1 [JSR234]		This API extends JSR 135 to provide more advanced multimedia capabilities.		M	M
	Music Capability	Not part of MSA 2.			
	3D Audio Capability			C	C
	Image Encoding Capability			M	M
	Image Post-Processing Capability			M	M
	Camera Capability			C	C

Specification	Optional Packages or Features	Comments	MSA 2 EP	MSA 2 SP	MSA 2 AP
	Tuner Capability			C	C
JSR 238 – Mobile Internationalization API Version 1.0 [JSR238]		This API enables the development of localized applications.	M	M	M
JSR 239 – Java Binding for the OpenGL ES API Version 1.0.1 [JSR239]		This API provides Java bindings to an OpenGL ES (Embedded Subset) 1.0 and 1.1 native 3D graphics library.		M	M
JSR 256 – Mobile Sensor API Version 1.1 [JSR256]		This API enables access to various sensors in the mobile device.	M	M	M
JSR 257 – Contactless Communication API Version 1.0 [JSR257]		This API enables the use of dedicated HW for using RFID and visual tags in applications.		C	C
JSR 258 – Mobile User Interface Customization API Version 1.0 [JSR258]		Provides a way to query and modify the user interface customization properties of a mobile device.		M	M
JSR 271 – Mobile Information Device Profile 3 Version 3.0 [JSR271]		This API is a fundamental component of MSA.		M	M
JSR 272 – Mobile Broadcast Service API for Handheld Terminals Version 1.0 [JSR272]		This API provides the ability to handle broadcast content, e.g. to view digital television and to utilize its features and services.		C	C
JSR 280 – XML API for Java ME Version 1.0 [JSR280]	**Core Package**	This package provides a general purpose XML parsing API.		M	M
	Events Package	Provides support for Document Object Model (DOM) Events and Views.		M	M
JSR 281 – IMS Services API Version 1.0 [JSR281]		This API provides an access to IMS services.			C
JSR 287 – Scalable 2D Vector Graphics API 2.0 for J2ME	**SVG Document API**	This API provides access to two dimensional (2D) Scalable Vector Graphics.		M	M

Specification	Optional Packages or Features	Comments	MSA 2 EP	MSA 2 SP	MSA 2 AP
Version 2.0 [JSR287]	**Advanced Vector Graphics API**	The Advanced Vector Graphics API supports high performance immediate mode rendering.		M	M
JSR 290 – Java Language & XML User Interface Markup Integration Version 1.0 [JSR290]		This JSR enables creation of Java ME applications combining Web UI markup technologies with Java code.			M
JSR 293 – Location API 2.0 Version 2.0 [JSR293]		This API provides a technology independent way to access location information.		C	C
JSR 297 – Mobile 3D Graphics API for J2ME Version 2.0 [JSR297]	**Core Block**	This API provides access to three dimensional (3D) graphics.		M	M
	Advanced Block	This API complements the Core Block by exposing the OpenGL ES 2.0 feature set.		C	C

Note: For any of the component JSRs, the version number denotes the required version of a respective configuration, profile or optional package (for example, Mobile 3D Graphics API 2.0). Backward compatible Maintenance Releases of any component JSRs MAY be used in compliant implementations. A later version of a configuration, profile, or optional package that includes new features MAY also be used in a compliant implementation, as long as the later version is backward compatible with the required version in this specification, and its inclusion is clearly noted in the documentation of the device.

Backward compatibility implies that applications that are written assuming the required version of the component JSR (without relying on any details specific to one particular implementation) are able to execute on the later version successfully without any changes to the application.

The Specification Lead and the EG of the component JSRs determine and declare (for example, by making a statement in the specification) whether the later version is backward compatible.

Note to Application Developers: Application developers should develop to the specific JSR versions and the clarifications defined in this specification.

6. Additional Clarifications (normative)

An MSA 2 compliant implementation MUST implement all Component JSRs that are mandatory in the implemented MSA 2 platform (EP, SP or AP). In addition, any number of other JSRs included in the MSA 2 specification MAY be implemented. Implementations of all these JSRs MUST comply with the clarifications outlined in the following component JSR Clarifications sections.

Some of the Component JSRs or their optional packages are conditionally mandatory. In those cases, an implementation claiming to be MSA 2 compliant MUST *conditionally* implement those component JSRs and their clarifications. Clarifications section for these component JSRs include a statement describing the *Condition for Inclusion* defining the condition or conditions on which the component JSR needs to be implemented as part of the MSA 2 platform (EP, SP or AP).

6.1 PDA Optional Packages for the J2ME Platform (JSR 75)

JSR 75 defines two independent optional packages, Personal Information Management and File Connection.

6.1.1 Rationale for Inclusion

PDA Optional Packages for the Java ME Platform contains two independent optional packages:

- *Personal Information Management (PIM):* This package gives Java ME applications access to personal information in address books, calendars, and to-do lists that are found on many mobile devices.
- *FileConnection (FC):* This package gives Java ME applications access to various forms of data (images, sounds, videos, and more) residing in file systems of mobile devices (including removable storage devices such as external memory cards).

6.1.2 Condition for Inclusion

None.

6.1.3 Clarifications for the PIM Optional Package

6.1.3.1 Support for ContactList, ToDoList, and EventList Is Mandatory

Clarification ID: CID.75.1

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Class `javax.microedition.pim.PIM`
  Method `javax.microedition.pim.PIM.openPIMList()`

- **Requirement Text:**

  An implementation of PIM APIs MUST support the following `PIMList` types:

  - `ContactList`
  - `ToDoList`
  - `EventList`

  All `PIMList` types MUST be supported in the following modes:

  - `READ_ONLY`
  - `READ_WRITE`
  - `WRITE_ONLY`

- **Justification/Notes:**

  This clarification defines mandatory `PIMList` types and modes, thus improving the predictability of implementations.

6.1.3.2 Mandatory ContactList Attributes

Clarification ID: CID.75.2

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Method `javax.microedition.pim.PIMList.getSupportedAttributes()`
  Method `javax.microedition.pim.PIMList.getAttributeLabel()`
  Method `javax.microedition.pim.PIMList.isSupportedAttribute()`
  Method `javax.microedition.pim.PIMItem.getAttributes()`

- **Requirement Text:**

  `ContactList` MUST support the following attributes：

  - `ATTR_FAX`
  - `ATTR_HOME`
  - `ATTR_MOBILE`
  - `ATTR_PREFERRED`
  - `ATTR_WORK`

  The fields MUST be mapped to the attributes in the following way:

Field	Attribute
TEL	ATTR_FAX, ATTR_HOME, ATTR_MOBILE, ATTR_PREFERRED, ATTR_WORK

Method PIMList.getSupportedAttributes() MUST NOT return ATTR_NONE in the attribute array. ATTR_NONE is a special meta-attribute that is sometimes used as an input parameter to denote the lack of attributes.

- **Justification/Notes:**

  This clarification defines mandatory ContactList attributes, thus improving the predictability of implementations.

6.1.3.3 Mandatory ContactList Fields

Clarification ID: CID.75.3

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Class javax.microedition.pim.PIMList
  Method javax.microedition.pim.PIMList.isSupportedField()
  Method javax.microedition.pim.PIMList.getSupportedFields()
  Method javax.microedition.pim.PIMList.getFieldDataType()
  Method javax.microedition.pim.PIMList.maxValues()

- **Requirement Text:**

  At least one ContactList in the device MUST support the following fields:

Field	Associated Data Type
ADDR	PIMItem.STRING_ARRAY
EMAIL, NOTE, TEL, URL	PIMItem.STRING
PHOTO	PIMItem.BINARY

  The EMAIL field MUST support at least two data values (email addresses). In addition, any ContactList MUST support one of the following fields:

  - Contact.FORMATTED_NAME field
  - Contact.NAME field with at least Contact.NAME_FAMILY and Contact.NAME_GIVEN indices supported

- **Justification/Notes:**

  This clarification defines mandatory ContactList fields, thus improving the predictability of implementations.

6.1.3.4 Mandatory EventList Fields

Clarification ID: CID.75.4

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Class `javax.microedition.pim.PIM`
  Method `javax.microedition.pim.PIM.listPIMLists()`

  Class `javax.microedition.pim.PIMList`
  Method `javax.microedition.pim.PIMList.isSupportedField()`
  Method `javax.microedition.pim.PIMList.getSupportedFields()`
  Method `javax.microedition.pim.PIMList.getFieldDataType()`

- **Requirement Text:**

  `EventList` MUST support the following fields:

Field	Associated Data Type
`SUMMARY`	`PIMItem.STRING`
`START`	`PIMItem.DATE`

  In addition, the first `EventList` returned by the `PIM.listPIMLists(PIM.EVENT_LIST)` method MUST support all of the following fields:

Field	Associated Data Type
`LOCATION, SUMMARY`	`PIMItem.STRING`
`END, START`	`PIMItem.DATE`
`ALARM`	`PIMItem.INT`

- **Justification/Notes:**

  This clarification defines mandatory `EventList` fields, thus improving the predictability of implementations.

6.1.3.5 Mandatory ToDoList Fields

Clarification ID: CID.75.5

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Class `javax.microedition.pim.PIMList`
  Method `javax.microedition.pim.PIMList.isSupportedField()`

Method `javax.microedition.pim.PIMList.getSupportedFields()`
Method `javax.microedition.pim.PIMList.getFieldDataType()`

- **Requirement Text:**

  `ToDoList` MUST support the following fields:

Field	Associated Data Type
`SUMMARY`	`PIMItem.STRING`
`PRIORITY`	`PIMItem.INT`
`COMPLETION_DATE, DUE`	`PIMItem.DATE`
`COMPLETED`	`PIMItem.BOOLEAN`

- **Justification/Notes:**

  This clarification defines mandatory `ToDoList` fields, thus improving the predictability of implementations.

6.1.3.6 Order of Returned Contact Lists

Clarification ID: CID.75.6

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Class `javax.microedition.pim.PIM`
  Method `javax.microedition.pim.PIM.listPIMLists()`

- **Requirement Text:**

  An implementation of the `javax.microedition.pim.PIM.listPIMLists(PIM.CONTACT_LIST)` method MUST return contact lists in the following order:

  - Contact list from the default phone book
  - Contact list from the device's phone book
  - Contact list from the smart card (such as SIM, USIM, or RUIM) used to identify the customer in the mobile network
  - Any other contact lists

- **Justification/Notes:**

  This clarification defines the order in which contact lists are returned by the `PIM.listPIMLists(PIM.CONTACT_LIST)` method, thus improving the predictability of implementations.

6.1.3.7 Localization of PIM List Names, Field Names, and Attribute Names

Clarification ID: CID.75.7

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

- **Requirement Text:**

  List names, field labels, and attribute labels in an implementation of the PIM Optional Package MUST be the same as in the native PIM applications. That is, if these names and labels are localized in the native PIM applications, the PIM Optional Package implementation MUST also use the same localized values.

- **Justification/Notes:**

  The goal of this clarification is to ensure consistent user experience accross Java and native PIM applications.

6.1.3.8 Order for Lists, Fields, and Attributes

Clarification ID: CID.75.8

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Interface `javax.microedition.pim.PIMList`
  Method `javax.microedition.pim.PIMList.items()`
  Method `javax.microedition.pim.PIMList.getSupportedFields()`
  Method `javax.microedition.pim.PIMList.getSupportedAttributes()`

- **Requirement Text:**

  Items in an `Enumeration` returned by `PIMList.items()` methods MUST be in the same order in which they are presented to the user by the User Interface of the native PIM application.

  Fields in an array returned by the `PIMList.getSupportedFields()` method MUST be in the same order in which they are presented to the user by the User Interface of the PIM application.

  Attributes in an array returned by the `PIMList.getSupportedAttributes()` method MUST be in the same order in which they are presented to the user by the User Interface of the PIM application.

- **Justification/Notes:**

  The goal of this clarification is to ensure consistent user experience accross Java and native PIM applications.

6.1.3.9 RepeatRule Class Functionality Compared With vCalendar v1.0 Specification

Clarification ID: CID.75.9

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification tool documents.

  Class `javax.microedition.pim.RepeatRule`

- **Requirement Text:**

  The functionality available in the `javax.microedition.pim.RepeatRule` class is more restricted than the functionality available through the vCalendar 1.0 object. Namely, the `RepeatRule` class supports only a single repeat rule and list of exception dates.

- **Justification/Notes:**

  An imported vCalendar 1.0 object allows for multiple repeat rules and complex exception criteria that is not supported by the `RepeatRule` class. The `RepeatRule` class only supports a single repeat rule and list of exception dates.

6.1.3.10 ContactList on SIM Card

Clarification ID: MSA2.075.002

- **Applicable Document, Section, Classes, and Methods:**

  PIM Optional Package Specification.

  Class `javax.microedition.pim.PIM`
  Method `javax.microedition.pim.PIM.listPIMLists()`
  Method `javax.microedition.pim.PIM.openPIMList(int, int)`
  Method `javax.microedition.pim.PIM.openPIMList(int, int, String)`

- **Requirement Text:**

  Phonebook contacts stored on SIM cards in the device MUST be exposed as PIM Optional Package ContactLists, one for each SIM card containing phonebook contacts.

- **Justification/Notes:**

  This requirement ensures that devices with SIM cards allow Java applications to access the `ContactList` on the SIM card. This improves the predictability of implementations.

6.1.4 Clarifications for the FileConnection Optional Package

6.1.4.1 Mark/Reset Functionality in Implementations of InputStream and DataInputStream

Clarification ID: CID.75.10

- **Applicable Document, Section, Classes, and Methods:**

FileConnection Optional Package Specification.

Interface `javax.microedition.io.file.FileConnection`
Method `javax.microedition.io.file.FileConnection.openInputStream()`
Method `javax.microedition.io.file.FileConnection.openDataInputStream()`

- **Requirement Text:**

Implementations of `InputStream` and `DataInputStream` interfaces (returned from the `FileConnection.openInputStream()` and `FileConnection.openDataInputStream()` methods respectively) MUST support mark/reset functionality defined in the description of these interfaces in the CLDC specification.

- **Justification/Notes:**

The clarification follows the same concept as outlined in the corresponding clarification for JSR 139 (CLDC 1.1) titled “Mark/Reset Functionality in InputStream Returned from Class.getResourceAsStream()”.

6.1.4.2 MIDlet Suite’s Storage Directory

Clarification ID: CID.75.11

- **Applicable Document, Section, Classes, and Methods:**

FileConnection Optional Package Specification.

- **Requirement Text:**

Each MIDlet suite in the device MUST be assigned a new directory where the suite’s MIDlets can store their data. This directory MUST be created when the MIDlet suite is installed on the device and deleted when the MIDlet suite is removed. If a MIDlet suite is removed and the directory is not empty, the implementation SHOULD make the user aware of the fact that the data in the directory will be deleted together with the MIDlet suite and SHOULD give the user an option to cancel the deletion.

During a MIDlet suite update, a new version of a MIDlet suite can retain data from the directory of an older version of a MIDlet suite. This data MUST be retained if RMS record stores are retained. In other words, MIDP rules on the inheritance of RMS record stores (pages 22-23 of the MIDP 2.1 specification, pages 35-36 of the MIDP 3.0 specification) MUST also be followed for data from the directory assigned to the MIDlet suite.

MIDlets from a MIDlet suite can find the name of their storage directory by querying the `fileconn.dir.private` system property. Localized names can be obtained from the `fileconn.dir.private.name` system property. See JSR 75 clarification "Directory Locator System Properties" for more details on these system properties.

Since the implementation is allowed to change the directory assigned to the MIDlet suite (this involves moving the data stored in the directory), MIDlets from the suite SHOULD query the properties each time before accessing the directory.

MIDlets belonging to other MIDlet suites MUST NOT have access to the MIDlet suites directory. In addition, the implementation SHOULD make its best effort to ensure that only MIDlets belonging to the MIDlet suite have access to the directory. However, MIDlets from a MIDlet suite SHOULD NOT assume that they have exclusive or secure access to the directory because in some implementations other native applications might still be able to access the data stored in the directory. In general, no security guarantees protect the data in the directory.

- **Justification/Notes:**

  This clarification addresses the need for a special directory where MIDlets belonging to the same MIDlet suite can store their files.

6.1.4.3 Minimum Supported Length of Pathname

Clarification ID: CID.75.12

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification.

- **Requirement Text:**

  An implementation MUST support full pathnames that are at least 255 Unicode characters long. This requirement defines the minimum supported length of a native pathname of a file in a file system, not the length of a URL that points to this file.

- **Justification/Notes:**

  This clarification defines a minimum supported length of full pathnames, thus improving the predictability of implementations.

6.1.4.4 Unescaping of URLs Presented to the User

Clarification ID: CID.75.13

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification.

- **Requirement Text:**

  When a User Interface element (for example, a security prompt) with a file URL (or part of it) needs to be shown to the user, an implementation MUST perform the necessary unescaping of escaped file URLs before presenting the UI element to the user.

- **Justification/Notes:**

  This clarification prevents malicious applications from misleading the user about the real name of a file. A malicious application could attempt to mislead the user with a file URL that contains escaped sequences (such as %2R%2E). For example, `file:///%66%69%6C%65%2E%74%78%74` could be used to represent `file:///file.txt`.

6.1.4.5 Directory Locator System Properties

Clarification ID: CID.75.14, MSA2.075.001

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification.

- **Requirement Text:**

  The following table contains names and descriptions of directory locator system properties that MUST be available to MIDlets:

System Property	Definition
`fileconn.dir.photos`	The URL of the default storage directory for photos captured with the integrated camera and other images.
`fileconn.dir.videos`	The URL of the default storage directory for video clips captured with the integrated camera or for downloaded and saved video clips.
`fileconn.dir.graphics`	The URL of the default storage directory for clip art graphics (caller group icons, background pictures, and other similar items).
`fileconn.dir.tones`	The URL of the default storage directory for ring tones and other related audio files.
`fileconn.dir.music`	The URL of the default storage directory for music files (MP3, AAC, and others).
`fileconn.dir.recordings`	The URL of the default storage directory for voice recordings made with the device.
`fileconn.dir.private`	The URL of the MIDlet suite's storage directory.

System Property	Definition
`fileconn.dir.roots.external`	The file system roots on external memory devices, such as memory cards. The system property MUST contain a concatenation of the external file system roots as returned from method `FileSystemRegistry.listRoots()`. The roots MUST appear in the same order as returned from this method. Roots MUST be separated by semicolon (`;`) characters in the concatenation. There MUST NOT be a semicolon after the last root. The concatenation MUST contain only the roots and the semicolons. Additional characters, such as whitespace, MUST NOT be added. Semicolon (`;`) characters in roots themselves MUST be escaped using the escape sequence mechanism defined in RFC 3986 [RFC3986] for reserved and non-ASCII characters in URIs. It is RECOMMENDED to avoid having semicolons in roots if possible. If there are no external file system roots, the system property MUST return `null` as its value. Device internal roots can be obtained by selecting those roots that are returned from method `FileSystemRegistry.listRoots()` but which do not appear in the list of external roots.

Property values differ from one implementation to another, but each non-null value MUST be a file URL that points to a specific directory in the file system of the device. If the device does not have a storage directory indicated by the system property, the value returned MUST be null. Two or more properties from the table above can map to the same directory and have exactly the same file URL. The implementation can change the values of all properties in this table dynamically (that is, during the execution of a MIDlet). This is REQUIRED to ensure proper handling of situations where the user, for example, changes a default directory through a native UI. Also, the properties in this table are accessible to all installed MIDlets, regardless of the protection domain. At the same time, the security policy of the device MAY restrict access to certain folders from MIDlets authenticated to a certain protection domain. A MIDlet can therefore be denied access to a directory indicated by a system property.

Directory names can be localized, that is, they can vary according to the user interface language. For this purpose, the following table provides system properties that can be used by MIDlets to get localized names for the directories, which can be shown in the user interface. The properties from this table MUST be available to MIDlets. `FileConnection` API implementations MUST define these properties (that is, return a non-null value) if they also define the directory URL properties in the previous table.

System Property	Definition
`fileconn.dir.photos.name`	Localized name of directory corresponding to the system property `fileconn.dir.photos`.
`fileconn.dir.videos.name`	Localized name of directory corresponding to the system property `fileconn.dir.videos`.
`fileconn.dir.graphics.name`	Localized name of directory corresponding to the system property `fileconn.dir.graphics`.
`fileconn.dir.tones.name`	Localized name of directory corresponding to the system property `fileconn.dir.tones`.
`fileconn.dir.music.name`	Localized name of directory corresponding to the system property `fileconn.dir.music`.
`fileconn.dir.recordings.name`	Localized name of directory corresponding to the system property `fileconn.dir.recordings`.
`fileconn.dir.roots.names`	Concatenation of localized names corresponding to roots returned by the `FileSystemRegistry.listRoots()` method. One localized name corresponds to each root returned by the method. Localized names MUST be in the same order as returned by the method. Root names MUST be separated by semicolon (`;`) characters in the concatenation. There MUST NOT be a semicolon after the last root name. The concatenation MUST contain only the root names and the semicolons. Additional characters, such as whitespace, MUST NOT be added. If no localized name exists for the root, the non-localized (logical) name is returned in the property for this root. Root names returned through this property MUST NOT contain the semicolon (`;`) character.
`fileconn.dir.private.name`	Localized name of directory corresponding to the system property `fileconn.dir.private`.

Also in this case, the implementation can dynamically change values of all properties in this table (that is, during the execution of a MIDlet).

- **Justification/Notes:**

  This clarification addresses the need to provide application developers with a method to learn about locations of common storage areas in the file system.

6.1.4.6 Unescaping of URLs by FileConnection.setFileConnection() Method

Clarification ID: CID.75.15

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification.

  Method `javax.microedition.io.file.FileConnection.setFileConnection()`

- **Requirement Text:**

  An implementation of the `FileConnection.setFileConnection()` method MUST NOT unescape the `fileName` argument. This is needed to allow MIDlets to open files and directories containing names with embedded escaped sequences (for example, `My%20Doc.txt`).

- **Justification/Notes:**

  The file system of the device might have files that have escape sequences literally embedded in their names (`My%20Doc.txt` compared with `My Doc.txt`). This clarification addresses the need for a method that MIDlets can use to open files with such names.

6.1.4.7 Types of Memory and Memory Cards Supported by FileConnection API

Clarification ID: CID.75.16

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification.

- **Requirement Text:**

  FileConnection APIs specification lists several memory card formats that could be supported by the implementation. A compliant implementation MUST support access to all memory cards (with file systems) that are available on the device.

- **Justification/Notes:**

  This clarification improves the predictability of implementations.

6.1.4.8 Throwing Correct Exception When Accessing ".." From FileConnection Root

Clarification ID: CID.75.17

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification.

  Method `javax.microedition.io.file.FileConnection.setFileConnection()`

- **Requirement Text:**

  If a `FileConnection` instance is connected to the file system's root and the parameter "`..`" is passed to the `setFileConnection()` method, `java.io.IOException` MUST be thrown.

- **Justification/Notes:**

  The specification for the `setFileConnection()` method does not specify what exception should be thrown when the current file connection is connected to the file system's root and the parameter "`..`" is passed to the `setFileConnection()` method. This clarification removes the ambiguity.

6.1.4.9 Typographical Errors in Security Section of FileConnection Interface Specification

Clarification ID: CID.75.18

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification, "Security" section of the `FileConnection` interface specification.

  Interface `javax.microedition.io.file.FileConnection`

- **Requirement Text:**

  The sentence:

  > "All three connections modes (READ_WRITE; WRITE_ONLY; and READ_ONLY) are supported for a file connection and determine the access requested from the security model."

  MUST be interpreted as:

  > "All three connection modes (READ_WRITE; WRITE; and READ) are supported for a file connection and determine the access requested from the security model."

- **Justification/Notes:**

  This clarification corrects typographical errors in the "Security" section of the `FileConnection` interface specification.

6.1.4.10 Access to File Attributes with canRead, canWrite and isHidden

Clarification ID: MSA2.075.004

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification

Interface `javax.microedition.io.file.FileConnection`
Method `setReadable(boolean readable)`
Method `setWritable(boolean writeable)`
Method `setHidden(boolean hidden)`
Method `canRead()`
Method `canWrite()`
Method `isHidden()`

- **Requirement Text:**

  In the `setReadable` method description the phrase "`canRead()` always returns true" is in conflict with the specification of the `canRead` method. The `setReadable` method description should read: "`canRead()` always returns true only if the attribute is not supported by the filesystem."

  In the `setWriteable` method description the phrase "`canWrite()` always returns true" is in conflict with the specification of the `canWrite` method. The `setWriteable` method description should read: "`canWrite()` always returns true only if the attribute is not supported by the filesystem."

  In the `setHidden` method description the phrase "`isHidden()` always returns false" is in conflict with the specification of the `canRead` method. The `setReadable` method description should read: "`isHidden()` always returns false only if the attribute is not supported by the filesystem."

- **Justification/Notes:**

  The description of the `canRead`, `canWrite`, and `isHidden` methods are specified to return the respective attributes of the file from the filesystem except in the case where the filesystem does not support the associated attribute.

  The `setReadable`, `setWritable`, and `setHidden` methods have no effect if the file system does not support settable attributes. However, in this case the methods require the respective `canRead`, `canWrite`, and `isHidden` methods to return fixed values in contradiction to the condition identified in those methods. The contradiction is resolved as described in the requirement text.

6.1.4.11 Default Values for File Attributes

Clarification ID: MSA2.075.005

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification

  Interface `javax.microedition.io.file.FileConnection`
  Method `create()`
  Method `mkdir()`

- **Requirement Text:**

  Files and directories created with methods `create()` and `mkdir()` MUST get the following default values for file and directory attributes:

  - `isHidden()` MUST return `false`, unless the file or directory name implies a hidden file or directory. For example, files and directories with names beginning with a period character ('.') are considered to be hidden in many Unix based operating systems.
  - `isReadable()` MUST return `true`, unless the file was created to a write only file system.
  - `isWritable()` MUST return `true.`

- **Justification/Notes:**

  The description of methods `create()` and `mkdir()` does not define default values for file and directory attributes. This causes problems when different device implementations use different defaults. For example, the same application running in two devices might create visible files in one and hidden files in another. This specification clarification reduces these differences between implementations.

6.1.4.12 File Attributes and Method rename()

Clarification ID: MSA2.075.006

- **Applicable Document, Section, Classes, and Methods:**

  FileConnection Optional Package Specification

  Interface `javax.microedition.io.file.FileConnection`
  Method `rename(java.lang.String newName)`

- **Requirement Text:**

  Calling method `rename()` MUST NOT change the file and directory attribute values (hidden, readable, writable), unless the new name implies that. For example, files and directories with names beginning with a period character ('.') are considered to be hidden in many Unix based operating systems.

- **Justification/Notes:**

  The description of method `rename()`does not define what should happen to file and directory attributes in a rename operation. This causes problems when some device implementations keep the attribute values, while others reset the values back to defaults. This specification clarification reduces these differences between implementations.

6.2 Java APIs for Bluetooth (JSR 82)

Bluetooth wireless technology (BT) is a widely used standard for wireless communication between devices. The JSR 82 specification defines a set of APIs that allow Java applications to use Bluetooth wireless technology.

6.2.1 Rationale for Inclusion

The JSR 82 Bluetooth API provides developers a mechanism to create applications and services using Bluetooth wireless technology. OBEX protocol is also widely used in mobile devices, and the JSR 82 OBEX API can be used to exchange objects such as files, pictures, calendar entries and business cards.

6.2.2 Condition for Inclusion

Conditions for inclusion of JSR 82 Bluetooth API and OBEX API are as follows:

- If the device supports Bluetooth wireless technology, the Bluetooth API MUST be supported.
- If the device supports the Bluetooth wireless technology, OBEX over Bluetooth (`btgoep://` URL prefix) and the OBEX API MUST be supported.
- If the device supports Infrared, OBEX over Infrared (`irdaobex://` URL prefix) and the OBEX API MAY be supported.
- If the device supports TCP/IP sockets, OBEX over TCP (`tcpobex://`) and the OBEX API MAY be supported.

6.2.3 Clarifications

6.2.3.1 Recommendation for Developers Not to Use ReceiveMTU and TransmitMTU

Clarification ID: CID.82.1

- **Applicable Document, Section, Classes, and Methods:**

  Java APIs for Bluetooth Specification v1.1.1.

- **Requirement Text:**

  When opening an L2CAP connection, an application SHOULD NOT use optional parameters `ReceiveMTU` and `TransmitMTU`. The use of these parameters significantly increases the risk of the connection establishment process ending in a failure.

  If an application requires an L2CAP connection with certain values of `ReceiveMTU` and `TransmitMTU` parameters, an application SHOULD act as follows:

  - Try to establish a connection not using `ReceiveMTU` and `TransmitMTU` parameters in the connection URL.
  - After the connection is established, check the values of the parameters. If the values are unsatisfactory, terminate the connection.

- **Justification/Notes:**

  The JSR 82 specification defines two optional parameters that can be used by applications when establishing L2CAP connections: `ReceiveMTU` and `TransmitMTU`. If an application uses these parameters, the risk of failure in connection establishment increases significantly, due to complex negotiation rules defined in the JSR 82 specification; and also because the semantics of the parameters are specific to the JSR 82 Bluetooth API. These new semantics can cause problems when the Bluetooth API is implemented on top of a standard Bluetooth stack. Therefore, this clarification recommends the developers not to use `ReceiveMTU` and `TransmitMTU` during connection establishment.

6.2.3.2 Bluetooth Power-Save Mode

Clarification ID: CID.82.2

- **Applicable Document, Section, Classes, and Methods:**

  Java APIs for Bluetooth Specification v1.1.1.

- **Requirement Text:**

  Some Bluetooth implementations use certain power-saving measures to avoid unnecessary battery drain. It is therefore possible that the Bluetooth stack expects regular activity on Bluetooth connections to keep them open.

  The developer SHOULD take this into account when developing applications that communicate irregularly via Bluetooth (for example, turn-based games) and either be prepared to handle an exception or keep the connection active with a “heartbeat” mechanism.

- **Justification/Notes:**

  This clarification helps create a predictable application environment by warning the developers that some Bluetooth stacks can go into a "power-save mode" and suggesting a solution to mitigate this risk.

It should be noted that preventing the device from entering the power-save mode will cause the battery to drain faster.

6.3 Mobile Media API (JSR 135)

This small-footprint API allows easy and simple access to and control of basic audio and multimedia resources and also addresses scalability and support for more sophisticated features.

6.3.1 Rationale for Inclusion

Multimedia capabilities are an essential feature in creating compelling, rich mobile applications. This API provides developers with capabilities for playback and capture of multimedia content.

6.3.2 Condition for Inclusion

None.

6.3.3 Clarifications

6.3.3.1 MIDI Content Format Support

Clarification ID: CID.135.1

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

  Method `javax.microedition.media.Manager.createPlayer()`.

- **Requirement Text:**

  Implementation MUST support the playback of Musical Instrument Digital Interface (MIDI) format content. This requirement concerns an implementation of the `Manager.createPlayer()` method.

  The MIDI implementation MUST support SP-MIDI [SP-MIDI_1] and the SP-MIDI 5-to-24 note profile for 3GPP specifications [SP-MIDI_2].

- **Justification/Notes:**

  JSR 135 specification does not require support for MIDI and also does not require the SP-MIDI implementation. This clarification mandates the support for the mentioned features and thus improves the predictability of implementations.

6.3.3.2 Support of AMR Content Format

Clarification ID: CID.135.2

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification. This clarification applies to the implementation of AMR content format for files that are provided as input to method `javax.microedition.media.Manager.createPlayer`.

- **Requirement Text:**

  Implementations of MSA 2 SP and MSA 2 AP MUST support the 3GPP Adaptive Multi Rate, Narrowband (AMR-NB) [AMR] content format for playback of sampled audio.

  Implementations of MSA 2 EP MUST support the 3GPP Adaptive Multi Rate, Narrowband (AMR-NB) [AMR] content format for playback of sampled audio if the device has a native player for the format.

  If AMR is supported, all the bitrates required by the AMR-NB specification MUST be supported.

  If the 3D Audio Capability of JSR 234 is supported as part of an implementation (this requirement does not apply to implementations supporting only MSA 2 EP), the method `javax.microedition.amms.SoundSource3D.addPlayer` MUST support adding `Player`s for AMR content format.

  However, AMR support is NOT REQUIRED for compliant *development tools*, *device emulators*, or a *reference implementation* running on a device emulator. Such implementations MUST support the initiation of AMR content playback in the API, but it is sufficient to only simulate the behavior of an AMR player. No actual sound needs to be generated, or if it is generated for simulation purposes, it does not need to play the actual content provided. The details related to such simulation are implementation dependent.

- **Justification/Notes:**

  To provide consistent support for a more efficient audio coding format, MSA defines support for AMR-NB content format.

6.3.3.3 PlayerListener Events

Clarification ID: CID.135.3

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

  Interface `javax.microedition.media.Player`
  Interface `javax.microedition.media.PlayerListener`

- **Requirement Text:**

  Additional requirements concerning `PlayerListener` events are as follows:

  - All `Players` MUST support the CLOSED, END_OF_MEDIA, ERROR, STARTED and STOPPED events.
  - `Players` for media types with an unknown duration at the start of the playback MUST support the DURATION_UPDATED event.
  - All `VolumeControl` implementations MUST support the VOLUME_CHANGED event.
  - All `VideoControl` implementations MUST support the SIZE_CHANGED event.
  - All `RecordControl` implementations MUST support the RECORD_ERROR, RECORD_STARTED and RECORD_STOPPED events.
  - All `StopTimeControl` implementations MUST support the STOPPED_AT_TIME event.

  All the events mentioned above MUST be delivered to registered `PlayerListeners.`

- **Justification/Notes:**

  JSR 135 specification does not specify which events need to be supported by various `Players`. This clarification mandates a set of events to be supported by all implementations.

6.3.3.4 Media Playback and Paused State of MIDP Application Model

Clarification ID: CID.135.4

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

  Interface `javax.microedition.media.Player`

- **Requirement Text:**

  In the MIDP 2.1 application model, the state of a MIDlet MAY be changed to Paused.

  If a `Player` is playing some media, the state of the `Player` MUST NOT be automatically altered by the implementation when the state of the MIDlet is changed.

  However, the implementation is allowed to mute the audio output, if it is conflicting with any other use of the device. An ongoing telephone call is one example of such a conflict. Muting the audio output MUST NOT affect the state of the `Player`. This requirement applies to Mobile Media API implementations irrespective of the underlying MIDP version.

- **Justification/Notes:**

  This clarification explains certain aspects of the relationship between the behavior of `Players` and the MIDlet Paused state in MIDP 2.1.

6.3.3.5 Full-Screen Behavior of Visual Players

Clarification ID: CID.135.5

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

  Interface `javax.microedition.media.VideoControl`

  Method `javax.microedition.media.VideoControl.setDisplayLocation()`
  Method `javax.microedition.media.VideoControl.setDisplaySize()`
  Method `javax.microedition.media.VideoControl.setDisplayFullScreen()`

- **Requirement Text:**

  The `VideoControl` interface includes methods `setDisplayLocation()`, `setDisplaySize()` and `setDisplayFullScreen()`.

  If the device supports playback of visual content with MMAPI, the implementation MUST support full-screen playback for visual content.

  In full-screen mode, the media content is scaled to a size as large as possible, given the constraints of the display, the scaling capability of the implementation for the given media content, and possibly other constraints of the device's user interface. The aspect ratio of the original content MUST be preserved. If the media content does not fill the entire display (due to the aspect ratio difference or implementation capability in scaling), it is implementation dependent how the remaining area of the display is filled. It is possible that even in full-screen mode the display contains some device specific graphics in addition to the rendered media content.

  If the display is set to full-screen mode using `setDisplayFullScreen()`, calling `setDisplayLocation()` and `setDisplaySize()` MUST NOT affect the full-screen presentation of the video. However, the parameters set in those method calls MUST be stored by the implementation, and the most recently set values MUST take effect if the video is brought into non-full-screen presentation by calling `setDisplayFullScreen(false)`.

- **Justification/Notes:**

  In the JSR 135 specification the definition of the interaction between the full-screen mode and the positioning and size parameters is ambiguous. This clarification removes that ambiguity.

6.3.3.6 VolumeControl Support by Players Producing Audio Is Mandatory

Clarification ID: CID.135.6

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

  Interface `javax.microedition.media.Controllable`

  Method `javax.microedition.media.Controllable.getControl()`

- **Requirement Text:**

  All `Players` producing audio output MUST support `VolumeControl`.

- **Justification/Notes:**

  The JSR 135 specification does not mandate support for any specific `Controls`. This clarification mandates support for the `VolumeControl` for all `Players` producing audio output, thus improving the predictability of implementations.

6.3.3.7 Consistency of Volume Control

Clarification ID: MSA2.135.001

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

  Interface `javax.microedition.media.control.VolumeControl`

- **Requirement Text:**

  The application can control the volume of the output using the interface `javax.microedition.media.control.VolumeControl`. In addition to this, the device MAY provide to the end user a volume control, e.g. by having dedicated volume keys for dynamic control of the volume or some settings for static control of the volume. In addition, the device may have a way to mute the audio output, e.g. in a silent mode.

  These settings MUST be implemented in such a way that they are equivalent to independent faders/switches representing these controls connected in series. The level used in the `VolumeControl` MUST NOT change when the end user changes the possible other volume controls, but they are independent controls connected in series.

  If the device provides a volume control to the end user, these two volume controls MUST behave in such a way that they are equivalent to having two faders representing these controls connected in series.

  The output level MUST be calibrated so that at the volume level setting of `80` in the `VolumeControl`, the output volume is similar as the device normally outputs in other

contexts. The application is able to increase this volume slightly by setting a level from `81` to `100` and decrease it by setting a level from `79` to `0`.

If the implementation does not provide a volume control to the end user, the level `80` in the `VolumeControl` MUST be calibrated so that the output volume is similar as the device normally outputs in other contexts.

In a similar way, the device mute control of the audio output MUST be independent from the `VolumeControl` `isMuted` and `setMute` methods. Changing the device mute control MUST NOT have an effect on the `isMuted` method and the `setMute` method MUST NOT be able to affect the device mute control. These behave as two independent switches on the audio path.

The same can be expressed using the following formula:

```
A = Java volume setting in VolumeControl (0…100)
B = System volume setting (0…100)
C = Java mute setting (0 = muted, 1 = not muted)
D = System mute setting (0 = muted, 1 = not muted)
E = Device specific maximum volume output level

Output_volume = min(1, (A/80)*(B/100))* C * D * E;
```

- **Justification/Notes:**

  This requirement improves the consistency of volume control implementations in different devices.

6.3.3.8 Camera Image Capture Resolution

Clarification ID: CID.135.7

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification. This clarification applies to implementation of camera capture functionality.

- **Requirement Text:**

  Implementations MUST support the same resolutions for image capture as supported by the system camera applications of the phone.

- **Justification/Notes:**

  Java applications should be able to use the same level of image capture capabilities as are available for the end user in the phone’s system camera applications.

  This applies to image capture functionality in both JSR 135 and JSR 234.

6.3.3.9 Support for Audio Mixing

Clarification ID: MSA2.135.010

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification.

- **Requirement Text:**

  If the device exposes audio mixing functionality to downloadable applications through a software API, system property `supports.mixing` MUST have value `true` and all the corresponding JSR 135 requirements MUST be met.

- **Justification/Notes:**

  Audio mixing is necessary for applications that require sound effects and background music to be played simultaneously.

  Note: Definition for the system property `supports.mixing` in the Mobile Media API specification lists the method calls that work when mixing is supported. Applications should be written to also handle the case when mixing is not supported, i.e. to handle the `MediaException` that may be thrown.

6.3.3.10 Support for Setting Media Time

Clarification ID: MSA2.135.005

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification

  Interface `javax.microedition.media.Player`
  Method `setMediaTime(long now)`

- **Requirement Text:**

  A compliant implementation MUST support setting the stream position before and during playback using the `Player.setMediaTime()` method according to the following:

  - `setMediaTime(0)` MUST be supported for all media types and protocols.
  - `setMediaTime()` with positive argument values MUST be supported for media types and file formats that support setting the media time and that are played back using protocols supporting a seeking function (file and rtsp) when `Player.getDuration()` returns a positive value. For a file connection, a granularity of 1 second MUST be supported in this case. For an RTSP connection, the given value must be sent using the RANGE attribute of the PLAY command to the RTSP server and the response must indicate the actual set time as received from the server [RFC2326].

- **Justification/Notes:**

  This requirement ensures that setting media time works with playback protocols that are able to support it.

6.3.3.11 Support for Media Duration

Clarification ID: MSA2.135.006

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification

  Interface `javax.microedition.media.Player`
  Method `getDuration()`

- **Requirement Text:**

  Implementations MUST return the real media duration from method `Player.getDuration()` when playing media using the FILE protocol. This means that implementations MUST NOT return `TIME_UNKNOWN` when playing media from a file.

  Implementations MUST return the real media duration from method `Player.getDuration()` when playing non-live media using the RTSP protocol. This means that implementations of RTSP media players MUST return the media duration as obtained from the RTSP protocol and `TIME_UNKNOWN` may only be returned when the information is not available (e.g. live streaming). The duration is retrieved from the attribute RANGE in the response to a DESCRIBE request. Implementations MUST support at least ranges defined in terms of Normal Play Time (NPT) units as defined in RFC 2326 [RFC2326].

- **Justification/Notes:**

  This requirement ensures that applications can rely on the duration information for non-live streams to be available if supported by the protocol (FILE and RTSP).

6.3.3.12 Permission for RTSP

Clarification ID: MSA2.135.012

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification

  Class `javax.microedition.media.Manager`
  Method `createPlayer(java.lang.String locator)`

- **Requirement Text:**

  Implementations of method `createPlayer(java.lang.String locator)` MUST check for permission `javax.microedition.io.Connector.rtsp` on MIDP 2.1 based devices. Implementations MUST throw `java.lang.SecurityException` if the application does not have the required permission.

  Implementations of method `createPlayer(java.lang.String locator)` MUST check for permission `javax.microedition.io.RTSPProtocolPermission` on MIDP 3 based devices. Implementations MUST throw `java.lang.SecurityException` if the application does not have the required permission.

  Definition of class `javax.microedition.io.RTSPProtocolPermission` <TBD>.

- **Justification/Notes:**

  Creating MMAPI Players to play back content using the RTSP protocol has been implemented in several devices, but there has been some confusion on what permission the applications should request to use RTSP as no dedicated RTSP permission has been defined. This clarification defines the permission for RTSP.

6.3.3.13 Media Format Support

Clarification ID: MSA2.135.007, MSA2.135.011

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification

  Interface `javax.microedition.media.Player`
  Method `createPlayer()`

- **Requirement Text:**

  Implementations MUST provide support for the media formats and media codings as specified in the table below.

Media Format Support in Mobile Media API (JSR 135)

				Platform support			
				EP		SP/AP	
File format	**Content**	**Coding**	**Supported level**	**Playback**	**Capture**	**Playback**	**Capture**
Tone sequence	audio			x		x	
MIDI	audio			x		x	
SP-MIDI	audio			x		x	
WAV	audio	8-bit, 8kHz, mono linear PCM		p		x	
AMR-NB	audio			p	p	x	x
AMR-WB	audio					x	x
MP3	audio	20 - 320 kbit/s @ 44.1kHz		x		x	
3GP	video	H.263	profile 0 level 10	p	c	x	c
			profile 3 level 45	p		x	
(any supported video can be combined with any supported audio)		MPEG-4 Part 2	VSP level 0b (1)	p		x	
			VSP level 3 (2)			x	
		H.264 (MPEG-4 Part 10)	Baseline level 1b (1)			x	
			Baseline level 1.2 (1)			x	
			Baseline level 1.3			x	
	audio	AMR-NB		p	c	x	x
		AMR-WB				x	
		AAC-LC				x	
		HE-AAC (AAC+ v2)				x	
MP4	video	MPEG-4 Part 2	VSP level 0b (1)			x	c
			VSP level 3 (2)			x	
(any supported video can be combined with any supported audio)		H.264 (MPEG-4 Part 10)	Baseline level 1b (1)			x	
			Baseline level 1.2 (1)			x	
			Baseline level 1.3			x	
	audio	AAC-LC				x	c
		HE-AAC (AAC+ v2)				x	
		MP3				x	

required
recommended

x = In all devices implementing the platform.
p = In devices that have a native player for the format.
c = In devices that have a camera.

(1) As defined in 3GPP Rel-6 packet switched streaming document: TS 26.234.
Rtsp availablity of a given codec subject to support of the related network speed in the device.

(2) As defined in 3GPP Rel-7 packet switched streaming document: TS 26.234.
Rtsp availablity of a given codec subject to support of the related network speed in the device.

Implementations MUST support the media types and codings marked with an ‘x’ on dark background in the table in the Playback and Capture columns for the implemented platform (EP or SP/AP).

Implementations MUST support the media types and codings marked with a ‘p’ on dark background in the table in the Playback column for the implemented platform (EP or SP/AP) if the device has a native player for the format.

Implementations MUST support the media types and codings marked with a ‘c’ on dark background in the table in the Capture column for the implemented platform (EP or SP/AP) if the device has a camera.

Corresponding recommendations for implementations are marked with symbols ‘x’, ‘p’ and ‘c’ on white background. Implementations are RECOMMENDED to support codings marked in this way in the same manner as the mandatory codings.

Combining any of the supported video codings with any of the supported audio codings in 3GP and MP4 container formats MUST be supported for playback.

Playback of 3GP and MP4 container formats containing audio only MUST be supported.

Capturing in 3GP and MP4 container formats MUST always produce files with both video and audio.

- **Justification/Notes:**

  This requirement ensures that applications can rely on a set of media formats and encodings to be supported.

6.3.3.14 MIME types and Controls for Media Playback

Clarification ID: MSA2.135.008, MSA2.135.009

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Media API Specification

  Interface `javax.microedition.media.Player`
  Method `createPlayer()`

- **Requirement Text:**

  Implementations that support a given media file format:

  - MUST recognize the MIME type(s) for the format as specified in the following table
  - MUST support the protocols specified for the format in the following table
  - MUST support the controls specified for the format in the following table

MIME types, Controls and protocols for supported file formats in Mobile Media API (JSR 135)

			Playback														
			Controls											Protocols			
	File format	MIME types	VolumeControl	StopTimeControl (1)	RateControl	PitchControl	MIDIControl	TempoControl	ToneControl	FramePositioningControl	GUIControl	VideoControl	MetaDataControl	device	file	http (4)	rtsp (2) (3)
Synthetic sound	Tone sequence	audio/x-tone-seq	x	x	x	x		x	x					x	x	x	
	MIDI	audio/midi, audio/mid	x	x	x	x	x	x						x	x	x	
	SP-MIDI	audio/sp-midi	x	x	x	x	x	x						x	x	x	
Sampled sound	WAV	audio/x-wav, audio/wav	x	x	x										x	x	
	AMR-NB	audio/amr	x	x											x	x	
	AMR-WB	audio/amr-wb	x	x											x	x	
	MP3	audio/mpeg, audio/mp3	x	x									x		x	x	
Container format	3GP (video+audio)	video/3gpp	x	x						x	x	x	x		x	x	x
	3GP (audio only)	audio/3gpp	x	x									x		x	x	x
	MP4 (video+audio)	video/mp4	x	x						x	x	x	x		x	x	x
	MP4 (audio only)	audio/mp4, audio/aac	x	x									x		x	x	x

x = supported

(1) StopTimeControl NOT required in Players for rtsp protocol.
(2) As defined in 3GPP Rel-6 packet switched streaming document: TS 26.234.
Rtsp availablity of a given codec subject to support of the related network speed in the device.
(3) As defined in 3GPP Rel-7 packet switched streaming document: TS 26.234.
Rtsp availablity of a given codec subject to support of the related network speed in the device.
(4) Streaming/Progressive HTTP must be supported in SP and AP.

Implementations on SP and AP MUST support streaming HTTP. This means that the media file playback is started before the whole file has been downloaded over HTTP.

6.4 Connected Limited Device Configuration (JSR 139) and Connected Device Configuration (JSR 218)

The MSA 2 Specification uses the Connected Limited Device Configuration version 1.1 (CLDC 1.1) [JSR139-1.1] as the required Java ME configuration for MSA 2 Entry Platform.

MSA 2 Standard Platform and MSA 2 Advanced Platform require Connected Limited Device Configuration version 1.1.1 (CLDC 1.1.1) [JSR139-1.1.1].

MSA 2 implementations MAY alternatively use CDC 1.2 [JSR218] instead of CLDC as defined in the specification for the Runtime-Execution-Environment attribute in MIDP 2.1 and MIDP 3.0 specifications.

A *configuration* of the Java ME platform specifies a subset of the Java programming language features and Java virtual machine features, as well as the core platform libraries, to support a wide range of consumer products.

6.4.1 Rationale for Inclusion

CLDC is a fundamental component of the Java ME platform.

6.4.2 Condition for Inclusion

None. A compliant implementation MAY use CDC instead of CLDC. For more information, see the MIDP 2.1 and MIDP 3.0 specification and the definition of the Runtime-Execution-Environment JAD / JAR manifest attribute.

6.4.3 Clarifications

6.4.3.1 Mark/Reset Functionality in InputStream Returned from Class.getResourceAsStream()

Clarification ID: CID.139.2

- **Applicable Document, Section, Classes, and Methods:**

  CLDC Specification version 1.1.1 and CDC Specification version 1.1.2

  Class `java.lang.Class`

  `public java.io.InputStream getResourceAsStream(String name)`

  Class `java.io.InputStream`

```
public boolean markSupported()
public synchronized void mark(int readlimit)
public synchronized void reset() throws IOException
```

- **Requirement Text:**

  The implementations of the `InputStream` interface returned from method `Class.getResourceAsStream()` MUST support the `mark/reset` functionality as defined in the description of this interface in the CLDC/CDC Specification.

- **Justification/Notes:**

  This requirement improves the interoperability between different CLDC/CDC implementations and supports more efficient memory management in applications that load large resources.

6.4.3.2 Output Format of System Property microedition.platform

Clarification ID: CID.139.3

- **Applicable Document, Section, Classes, and Methods:**

  CLDC Specification version 1.1.1, Section 6.2.12 “Property Support”, Table 1.

- **Requirement Text:**

  The value returned by the system property `microedition.platform` MUST follow the syntax described in the following table:

System Property	Explanation	Value
`microedition.platform`	Name of the host platform or device	`Manufacturer_name Device_model_number [“/”version_number ] [“/”additional_comments ]`

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

- **Justification/Notes:**

  In the CLDC Specification, the value returned by the system property `microedition.platform` is defined as implementation-dependent (see CLDC Specification version 1.1.1, Section 6.2.12, Table 1). Consequently, additional

requirements for the property value are needed to allow applications to precisely identify the type of the host device or platform.

6.4.3.3 System Property for MSA Version

Clarification ID: CID.139.4, MSA2.139.001

- **Applicable Document, Section, Classes, and Methods:**

  CLDC Specification, version 1.1/1.1.1, Section 6.2.10 “Property Support”.

- **Requirement Text:**

  Implementations MUST support the following system property (through the `System.getProperty()` method):

System Property	Explanation
`microedition.msa.version`	Describes the MSA 1 Specification the implementation is compatible with. The value MUST be: • `1.1` for MSA 2 implementations supporting all of the MSA 1.1 APIs or their later compatible versions. • `1.1-SUBSET` for MSA 2 implementations supporting the MSA 1.1 Subset of APIs or their later compatible versions, but not the full set of APIs. • `null` for all other MSA 2 implementations.

- **Justification/Notes:**

  This is a legacy system property defined in MSA 1.0. A set of system properties have been defined in MSA 2 to provide more accurate information about the different MSA platforms that the implementation is compatible with. Applications written for MSA 1 can query this legacy system property to learn whether the implementation is compliant with the MSA 1 Specification, and which version of the specification is supported. New applications should use the new system properties providing more detailed information.

6.4.3.4 System Properties for Detailed MSA Version

Clarification ID: MSA2.139.001

- **Applicable Document, Section, Classes, and Methods:**

  CLDC Specification, version 1.1/1.1.1, Section 6.2.10 “Property Support”.

- **Requirement Text:**

  Implementations MUST support the following system properties (through the `System.getProperty()` method):

System Property	Explanation
`microedition.msa.2.0.entry`	`true` for all MSA 2.0 implementations and later MSA versions that are backward compatible with the MSA 2.0 Entry Platform.
`microedition.msa.2.0.standard`	`true` for MSA 2.0 Standard Platform and MSA 2.0 Advanced Platform implementations and later MSA versions that are backward compatible with the MSA 2.0 Standard Platform. `null` for all other implementations.
`microedition.msa.2.0.advanced`	`true` for MSA 2.0 Advanced Platform implementations and later MSA versions that are backward compatible with the MSA 2.0 Advanced Platform.. `null` for all other implementations.

- **Justification/Notes:**

  New system properties will be defined for later MSA versions, but all new MSA implementations must still support the old ones. This way applications can simply check if the MSA platform and platform version the application relies on is supported by the implementation.

6.5 J2ME Web Services (JSR 172)

The J2ME Web Services Specification (JSR 172) defines the APIs for utilizing web services in Java ME client devices.

The JSR 172 specification consists of two optional packages that can be implemented independently:

- XML parser optional package (JAXP subset)
- Web services optional package (JAX-RPC subset)

6.5.1 Rationale for Inclusion

This API provides a standard way to support web services and XML parsing in a mobile device.

6.5.2 Condition for Inclusion

None.

6.5.3 Clarifications

6.5.3.1 System Property for JAXP Subset API Version

Clarification ID: CID.172.1

- **Applicable Document, Section, Classes, and Methods:**

  J2ME Web Services Specification, Section 2.8, “JAXP Subset APIs”.

- **Requirement Text:**

  An implementation of JAXP Subset APIs MUST support the following system property (through the CLDC `System.getProperty()` method):

System Property	Explanation
`xml.jaxp.subset.version`	Version of JAXP Subset APIs supported by the device, for example, `1.0`.

- **Justification/Notes:**

  Application developers can use this property to determine if JAXP Subset APIs are present on the device and the version of those APIs.

6.5.3.2 System Property for JAX-RPC Subset API Version

Clarification ID: CID.172.2

- **Applicable Document, Section, Classes, and Methods:**

  J2ME Web Services Specification, Chapter 7, “JAX-RPC Subset Core APIs”.

- **Requirement Text:**

  An implementation of JAX-RPC Subset APIs MUST support the following system property (through the CLDC `System.getProperty()` method):

System Property	Explanation
`xml.rpc.subset.version`	Version of JAX-RPC Subset APIs supported by the device, for example, `1.0`.

- **Justification/Notes:**

  Application developers can use this property to determine if JAX-RPC Subset APIs are present on the device and the version of those APIs.

6.6 Security and Trust Services API (JSR 177)

The Security and Trust Services API (SATSA) Specification (JSR 177) defines optional packages to specify APIs that provide security and trust services for mobile devices. JSR 177 addresses the following needs:

- Secure storage to protect sensitive data, such as the user's private keys, public key (root) certificates, service credentials, and personal information.
- Cryptographic operations to support payment protocols, data integrity, and data confidentiality.
- A secure execution environment to deploy custom security features.

Java ME applications can rely on these features to handle several value-added services, such as user identification and authentication, banking, payment, and loyalty applications.

The JSR 177 specification consists of four optional packages that can be implemented independently:

- The SATSA-APDU optional package defines an API to support communication with smart card applications using the APDU protocol.
- The SATSA-CRYPTO optional package defines a subset of the J2SE cryptography API. It provides basic cryptographic operations to support message digest, signature verification, encryption, and decryption.
- The SATSA-PKI optional package defines an API to support application-level digital signature signing (but not verification) and basic user credential management. To enable broader reuse, this API is independent of the types of security elements that are utilized by a Java ME device.
- The SATSA-JCRMI optional package defines a Java Card RMI client API that allows a Java ME application to invoke a method of a remote Java Card object.

6.6.1 Rationale for Inclusion

Security is an important element in a wide variety of mobile applications and services. This API provides a generic way to access security services provided by the underlying system.

6.6.2 Condition for Inclusion

JSR 177 MUST be implemented, depending on the following conditions:

- The SATSA-CRYPTO optional package is MANDATORY and MUST be supported.

- The SATSA-APDU optional package is CONDITIONALLY MANDATORY: If an applicable security element (such as a smart card) is present, the SATSA-APDU optional package MUST be supported.
- The SATSA-PKI optional package is CONDITIONALLY MANDATORY: If an applicable security element (such as a smart card) is present, the SATSA-PKI optional package MUST be supported.
- The SATSA-JCRMI optional package is NOT REQUIRED.

6.6.3 Clarifications

6.6.3.1 Recommended Cryptography Algorithms Are Mandatory

Clarification ID: CID.177.1

- **Applicable Document, Section, Classes, and Methods:**

  Security and Trust Services API Specification, Appendix E, “Recommended Algorithms for the SATSA-CRYPTO Optional Package”.

- **Requirement Text:**

  Implementations of the JSR 177 Crypto Optional Package MUST implement the recommended algorithms, algorithm modes, and padding schemes as described in Appendix E of the JSR 177 Specification.

- **Justification/Notes:**

  This clarification is based on the feedback received from several device manufacturers. Mandating the recommended practices in this area improves the compatibility and interoperability of JSR 177 implementations.

6.6.3.2 Access Control

Clarification ID: CID.177.2

- **Applicable Document, Section, Classes, and Methods:**

  Security and Trust Services API Specification, Appendix A, “Recommended Security Element Access Control”.

- **Requirement Text:**

  Implementations MUST define the necessary Access Control mechanisms. Implementations MUST follow the conventions defined in JSR 177, Appendix A, “Recommended Security Element Access Control”, unless these mechanisms are explicitly superseded by another comprehensive Access Control Policy.

- **Justification/Notes:**

  Access control is an essential part of JSR 177 security.

6.6.3.3 JSR 177 Security Permissions

Clarification ID: CID.177.3

- **Applicable Document, Section, Classes, and Methods:**

  Security and Trust Services API Specification, Appendix B, “Security Permissions”.

- **Requirement Text:**

  Implementations MUST define the necessary Security Permissions according to JSR 177, Appendix B, “Security Permissions” except where superseded by Table 6 of the “Security Requirements” section (“Assigning Permissions and API Calls Specified in the Security and Trust Services API to Function Groups”).

  The security policy MAY be superseded by another comprehensive security policy.

- **Justification/Notes:**

  Security permissions are an essential part of JSR 177 security.

6.6.3.4 System Property for the SATSA-CRYPTO API Version

Clarification ID: CID.177.4

- **Applicable Document, Section, Classes, and Methods:**

  Security and Trust Services API Specification, Chapter 2, “Package Summary”.

- **Requirement Text:**

  An implementation of the SATSA-CRYPTO API MUST support the following system property (through the CLDC `System.getProperty()` method):

System Property	Explanation
`microedition.satsa.crypto.version`	Version of the SATSA-CRYPTO API supported by the device. For example, `1.0.1`.

- **Justification/Notes:**

  Application developers can use this property to determine if the SATSA-CRYPTO API is present on a device, and the version of the API.

6.6.3.5 System property for the SATSA-APDU API version

Clarification ID: CID.177.5

- **Applicable Document, Section, Classes, and Methods:**

  Security and Trust Services API Specification, Chapter 2, “Package Summary”.

- **Requirement Text:**

  An implementation of the SATSA-APDU API MUST support the following system property (through the CLDC `System.getProperty()` method):

System Property	Explanation
`microedition.satsa.apdu.version`	Version of the SATSA-APDU API supported by the device. For example, `1.0.1`.

- **Justification/Notes:**

  Application developers can use this property to determine if the SATSA-APDU API is present on a device, and the version of the API.

6.6.3.6 System property for the SATSA-PKI API version

Clarification ID: CID.177.6

- **Applicable Document, Section, Classes, and Methods:**

  Security and Trust Services API Specification, Chapter 2, “Package Summary”.

- **Requirement Text:**

  An implementation of the SATSA-PKI API MUST support the following system property (through the CLDC `System.getProperty()` method):

System Property	Explanation
`microedition.satsa.pki.version`	Version of the SATSA-PKI API supported by the device. For example, `1.0.1`.

- **Justification/Notes:**

  Application developers can use this property to determine if the SATSA-PKI API is present on a device, and the version of the API.

6.7 SIP API for J2ME (JSR 180)

JSR 180 defines a multipurpose SIP API for Java ME clients. It enables SIP applications to be executed in memory-limited terminals, especially targeting mobile phones.

6.7.1 Rationale for Inclusion

JSR 180 enables applications to take advantage of the SIP protocol.

6.7.2 Condition for Inclusion

None.

6.7.3 Clarifications

None.

6.8 Wireless Messaging API (JSR 205)

JSR 205 API provides an interface to the messaging functionality of the device. It allows the device to send and receive messages in SMS and MMS formats and receive messages in CBS format.

6.8.1 Rationale for Inclusion

Messaging is a phone feature that must be available to Java application developers.

6.8.2 Condition for Inclusion

None.

6.8.3 Clarifications

6.8.3.1 Message Handling and Buffering

Clarification ID: CID.205.1

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification, Chapters D2.3 (page 62), B2.0 (page 53) and A2.3 (page 49).

- **Requirement Text:**

  An implementation can work in a way that a message addressed to an application is first fully downloaded to the device and dispatched to the application afterwards. If this is the case, the following requirements apply.

  A message addressed to an application (using a port number in the SMS case or an application ID in the MMS case) MUST be saved permanently if at least one of the following conditions is fulfilled:

  - The application is registered in the `PushRegistry` to receive this type of messages.
  - The application is running and an active `Listener` is registered to receive notifications about messages of this type.

  When the device has insufficient memory to buffer a new incoming message, the situation MUST be handled as follows:

- If some messages are already buffered for the target application and the total size of these messages is greater than the space needed for the new message, an appropriate number of these messages MUST be deleted (oldest messages MUST be deleted first), and the new message MUST be buffered. The number of messages to be deleted depends on the size of the new message.
- If there are no previously buffered messages for the target application, or if the total size of the previously buffered messages is less than the needed space for the new message, the new message MUST be discarded.

A buffered message addressed to an application MUST be deleted in the following cases:

- The application reads the message.
- The application is removed from the device.
- The application removes the connection related to the buffered message from the `PushRegistry`.

- **Justification/Notes:**

  This clarification enhances the predictability of the platform.

6.8.3.2 Removal of a Message With an Unrecognized Application ID

Clarification ID: CID.205.2

- **Applicable Document, Section, Classes, and Methods**

  Wireless Messaging API Specification. This clarification addresses ambiguities in Chapters D2.3 (page 62), B2.0 (page 53) and A2.3 (page 49) of JSR 205.

- **Requirement Text**

  If a device is receiving a message (SMS, MMS, CBS) with an unrecognized application ID or port number, the device MUST, with the exception of SMS TextMessage, immediately delete the message without any notification to the user. For an SMS TextMesage with an unrecognized port number, the device MUST delete the port number of the SMS message and handle it as an ordinary SMS TextMessage (i.e. the SMS message becomes an SMS message without a port number).

- **Justification/Notes**

  This clarification prevents flooding a device's inbox with messages that neither the user nor the application is expecting.

6.8.3.3 BCC Header

Clarification ID: CID.205.3

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification. This clarification addresses ambiguities in Chapter D1.1 of JSR 205 (page 59).

- **Requirement Text:**

  All devices MUST support the `bcc` field for `MultipartMessages`.

- **Justification/Notes:**

  Eliminating the optionality of the BCC header availability enhances the platform consistency.

6.8.3.4 Encoding in MessagePart

Clarification ID: CID.205.4

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification.

  Class `javax.wireless.messaging.MessagePart`

- **Requirement Text:**

  The `encoding` parameter in `MessagePart` objects MUST be mapped to the character set indicated by the `charset=` parameter in the `Content-Type` header. The `encoding` parameter in the `MessagePart` object does not affect the choice of `Content-Transfer-Encoding`, which is chosen automatically by the implementation.

- **Justification/Notes:**

  In the `MessagePart` class, the object contains an attribute called `encoding`. It is a bit unclear how it maps to the MMS transport, because the MMS transport has two concepts related to `encoding`: the character set used by the message body and the so-called `Content-Transfer-Encoding`.

6.8.3.5 Creation of Message Objects of Different Type than Used by the MessageConnection Instance

Clarification ID: CID.205.5

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification.

  Method `javax.wireless.messaging.MessageConnection.newMessage(String)`

- **Requirement Text:**

  When calling the `MessageConnection.newMessage` method, the method MUST throw `IllegalArgumentException` if it is called with a message type parameter that does not match the message types supported by the transport protocol with which this `MessageConnection` instance is associated.

  If the `MessageConnection` was opened with an `mms:` URI, the `newMessage` method only allows the creation of messages of type `MULTIPART_MESSAGE` and MUST throw `IllegalArgumentException` if called with message types `TEXT_MESSAGE` or `BINARY_MESSAGE`.

  If the `MessageConnection` was opened with an `sms:` URI, the `newMessage` method only allows the creation of messages of types `TEXT_MESSAGE` and `BINARY_MESSAGE` and MUST throw `IllegalArgumentException` if called with message type `MULTIPART_MESSAGE`.

- **Justification/Notes:**

  This clarification specifies exact exception behavior when a new message is created that does not match the transport protocol (SMS or MMS) specified when the message connection object was created. It also specifies exception behavior when an attempt is made to use an inappropriate type of message for a given transport.

6.8.3.6 Messaging Protocol Support

Clarification ID: CID.205.6

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification. Underlying protocol support for the API.

- **Requirement Text:**

  The MMS protocol MUST be supported as specified in Appendix D of the JSR 205 specification.

- **Justification/Notes:**

  JSR 205 does not mandate any particular underlying messaging protocols.

6.8.3.7 System property for Wireless Messaging API version

Clarification ID: CID.205.7

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification, Chapter 3, “Package javax.wireless.messaging”.

- **Requirement Text:**

  A Wireless Messaging API implementation MUST support the following system property (through the CLDC `System.getProperty()` method):

System Property	Explanation
`wireless.messaging.version`	Version of the Wireless Messaging API supported by the device. For example, `2.0`.

- **Justification/Notes:**

  Application developers can use this property to determine if the Wireless Messaging API is present on the device, and the version of the API.

6.8.3.8 System properties for SMSC and MMSC

Clarification ID: MSA2.205.002

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification, Appendix A, Table A-3: “Property Name and Description for SMSC Addresses”.

  Wireless Messaging API Specification, Appendix D, Table D-5: “Property Name and Description for MMSC Addresses”.

- **Requirement Text:**

  If SMSC address is unknown, due to no SIM card present or no SMSC set, system property `wireless.messaging.sms.smsc` MUST return null as its value.

  If MMSC address is unknown, due to no SIM card present or no MMSC set, system property `wireless.messaging.mms.mmsc` MUST return null as its value.

- **Justification/Notes:**

  "Wireless Messaging API version 2.0" specification does not define the expected bahavior in case of trying to retrieve SMSC and MMSC addresses when the addresses are unknown. This requirement makes implementations more predictable by defining the expected behavior.

6.8.3.9 Case Insensitivity of Address Type Fields

Clarification ID: MSA2.205.003

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification

  Interface `javax.wireless.messaging.MultipartMessage`
  Method `removeAddress(String, String)`

- **Requirement Text:**

  The implementation of method `removeAddress(String, String)` MUST treat type strings as case insensitive. This makes the behavior aligned with method `addAddress(String, String)`.

- **Justification/Notes:**

  "Wireless Messaging API version 2.0" specification for the method can be misinterpreted. This requirement makes implementations more predictable by clarifying the expected behavior.

6.8.3.10 Message Timestamp Value

Clarification ID: MSA2.205.001

- **Applicable Document, Section, Classes, and Methods:**

  Wireless Messaging API Specification

  Interface `javax.wireless.messaging.Message`
  Method `getTimestamp()`

  Interface `javax.wireless.messaging.MessageConnection`
  Method `newMessage(java.lang.String type)`
  Method `newMessage(java.lang.String type, java.lang.String address)`
  Method `receive()`

- **Requirement Text:**

  Method `getTimestamp()` MUST return `null` for Message objects created by the application using methods `MessageConnection.newMessage(java.lang.String type)` and `MessageConnection.newMessage(java.lang.String type, java.lang.String address)` both before and after sending the Message object with method `MessageConnection.send(Message msg)`.

  Method `getTimestamp()` MUST return the timestamp set by the network for Message objects received using method `MessageConnection.receive()`.

- **Justification/Notes:**

  "Wireless Messaging API version 2.0" specification defines the return value for the `getTimestamp()` method as "the timestamp indicating when this message has been sent.". This definition leaves some room for interpretation and may cause undesired differences between device implementations.

  For received messages, the timestamp is intended to reflect the timestamp value in the received protocol message. Due to the asynchronous nature of message sending, the `Message` object that was used in `MessageConnection.send(Message msg)` cannot be later modified to contain the actual time the message was sent. The method returns

immediately and only queues the message, whereas the actual message sending can happen later, for example, due to network unavailability.

6.9 Content Handler API (JSR 211)

The Content Handler API (CHAPI), defined by JSR 211, allows an application to invoke an appropriate application on the device (the handler application) based on the type of content being processed. The appropriate handler is launched by the application management system (AMS) of a device implementation, which selects a registered Java ME application (or non-Java application) based on the content type (or types) that it can handle. The content type is specified by the invoking application, or discovered by invoking a specified Uniform Resource Locator (URL). A content handler is any application that registers itself to handle a certain type of content. Integration into the AMS gives the user a seamless and natural transition between applications and content handlers.

6.9.1 Rationale for Inclusion

This API enables the launching of Java ME applications based on content type.

6.9.2 Condition for Inclusion

None.

6.9.3 Clarifications

6.9.3.1 Requirements for Preregistered Content Handlers

Clarification ID: MSA2.211.003, MSA2.211.006

- **Applicable Document, Section, Classes, and Methods:**

  Content Handler API Specification, sections “Installing Content Handlers” and "Content Handlers and the Mobile Information Device Profile",

- **Requirement Text:**

  All preregistered content handlers MUST be visible in the content handler registry.

  Preregistered content handlers for installing Java applications, such as MIDlet Suites, MUST be registered to handle file suffixes ".jad" and ".jar". This is required in addition to registering handlers for the respective MIME types as defined in the Content Handler API specification [JSR211].

  Preregistered content handlers for installing Java applications, such as MIDlet Suites, MUST be registered with action `ContentHandler.ACTION_INSTALL`. This means that it MUST be possible to invoke the installer handler both with `ACTION_INSTALL` and without any action.

- **Justification/Notes:**

  The Content Handler API specification leaves some details of preregistered content handlers undefined. These clarifications improve consistency between different device implementations.

6.9.3.2 Returning Status after Content Handler Invocations

Clarification ID: MSA2.211.002

- **Applicable Document, Section, Classes, and Methods:**

  Content Handler API Specification

- **Requirement Text:**

  If an invocation is made to handle a JAD file without action or with `ACTION_OPEN`, the user interaction flow MUST be the same as in a normal installation, thereby asking the user to launch the application after installation.

  If an invocation is made to handle a JAD file with `ACTION_INSTALL`, the user MUST not be prompted to launch the application after installation and the installed application MUST not be launched directly after install. Any static registrations like Push Registry or Content Handler registrations are however handled as part of the normal installation process.

  If a response has been requested by the calling application by calling `setResponseRequired(true)`, the caller MUST be notified of success or failure of the installation using “OK”, “ERROR” or “CANCELLED” responses after the installation attempt has finished. The status of the invocation will have the following meanings:

  - OK: Appliction has been installed successfully
  - ERROR: Application was not successfully installed. `Invocation.getArgs()` MUST return the number and text description of the installation status report (as defined in MIDP Provisioning).
  - CANCELLED: The request was cancelled by the user.

  If an application wishes to do a self update, it could invoke a JAD install on an updated JAD file using `ACTION_INSTALL` with `setResponseRequired(true)`. Since the application to be installed is the calling application itself and response required has been set, the updated application will be launched and it will receive a notification after the installation regardless of weather the update succeeded or failed - not as the installed application, but as the caller.

- **Justification/Notes:**

  Applications can use this procedure for installing handlers and external applications and be sure that they will be notified of the success or failure of the installation. If the calling application wishes to launch the handler after installation, it may do so by calling the handler using CHAPI.

This method also provides applications a convenient, straightforward and reliable way to update themselves and get restarted after the update.

6.9.3.3 Invocation Protocol Support

Clarification ID: MSA2.211.009

- **Applicable Document, Section, Classes, and Methods:**

  Content Handler API Specification

- **Requirement Text:**

  The JSR 211 implementation MUST support invocations through protocol schemes http:, https: and file:.

- **Justification/Notes:**

  The only protocol scheme required by JSR 211 is http. This requirement ensures that invocations can also be made for remote content that is accessed using https and content that is stored locally in the file system.

6.9.3.4 Handling parameters in MIME-types

Clarification ID: MSA2.211.005

- **Applicable Document, Section, Classes, and Methods:**

  Content Handler API Specification, Chapter 2: “Implementation Requirements”

  Package `javax.microedition.content`
  Class `Invocation`
  Method `findType()`

- **Requirement Text:**

  The CHAPI Implementation Requirements specify that scheme and protocol for RFC 2616 (HTTP) [RFC2616] must be supported. In addition, the content type specified by RFC 2616 in the Content-Type header uses only the type/subtype string as the CHAPI content type. Any additional parameters as defined by RFC 2046 [RFC2046] MUST be ignored.

  The `findType()` method MUST set the type of the content using type information found by accessing the content and ignoring parameters or other information that does not intrinsically identify the type of content. For example, the Content-Type header of HTTP includes type, subtype and parameters. Only the type/subtype string is used to set the type of the content in the `Invocation`.

- **Justification/Notes:**

  JSR 211 does not define how implementations should handle MIME types with parameters. For example, if there is a handler registered for type "text/vnd.sun.j2me.app-descriptor" and someone does an invocation for "text/vnd.sun.j2me.app-descriptor; charset=iso-8859-1", or the other way around.

  RFC 2045 [RFC2045] describes media type parameters as: "Parameters are modifiers of the media subtype, and as such do not fundamentally affect the nature of the content."

  The CHAPI defines types as opaque strings that are not case sensitive (See `Registry` class javadoc). MIME types are used as examples of types but the type strings are not defined to be interpreted as MIME-types.

  The gap in the specification is the mapping of the content type as defined by the RFC 2616 (HTTP specification) into the type identifiers used by CHAPI. The assumption is that the type/subtype from http is used for the type. This assumption needs to be specified (see Requirement Text).

  Note that the type is only used to dispatch to the content handler. The content handler itself has access to the Content-Type header via the `HttpConnection` and can use the Parameters to correctly decode the content.

6.9.3.5 Unregistering handler registrations after an application upgrade

Clarification ID: MSA2.211.008

- **Applicable Document, Section, Classes, and Methods:**

  Content Handler API Specification

  Package `javax.microedition.content`
  Class `Registry`
  Method `unregister(java.lang.String classname)`

- **Requirement Text:**

  The current application package refers to a newly installed application or an application that has been upgraded. In MIDP, for example, it is the MIDlet Suite that is installed and upgraded. Dynamic registrations are retained across the upgrade and remain registered in the current application. Method `unregister(java.lang.String classname)` MUST be able to remove all dynamic registrations including the ones that were retained over an application upgrade.

- **Justification/Notes:**

  Content Handler API specification defines that dynamic handler registrations are retained over application upgrades. It also defines that an application can only unregister handlers registered "in the current application package". This clarification defines the behavior when attempting to unregister handlers registered in a previous version of the current application package.

6.9.3.6 Selection Between Multiple Matching Handlers

Clarification ID: MSA2.211.010

- **Applicable Document, Section, Classes, and Methods:**

  Content Handler API Specification

  Package `javax.microedition.content`
  Class `Registry`
  Method `invoke(Invocation invocation)`
  Method `invoke(Invocation invocation, Invocation previous)`

- **Requirement Text:**

  The content handler API allows multiple handlers to register for the same content type. Although the specification allows applications to explicitly call a given handler, it also allows calls that match with multiple handlers. As a result of this, a call to `Registry.invoke()` may find multiple matching handlers and the implementation needs to choose the one to invoke. According to the specification, the logic to choose the content handler to be invoked is implementation dependent.

  As the way in which the handler is selected affects usability, we give a slightly stricter specification of that behaviour in this document.

  1. As a general rule, the device MUST provide a mechanism for the user to choose which of the multiple matching handers is invoked and not just arbitrarily select one for the user. This mechanism can be either a user prompt during the call to `Registry.invoke()` or a handler configuration that the user can make in advance to select which handler gets precedence when multiple handlers match. A possible implementation can also be a combination of the two.

  2. As an exception to the rule above, an implementation MAY give precedence to some prebuilt content handlers over later downloaded ones. This may be important for security and usability reasons. There may be content types that by default should be handled by the prebuilt handler and not by a 3rd party handler. It should be noted that all registered handlers can still be called. They just will not get selected when multiple handlers match the invocation.

  3. If prompting is used:

     a. The prompt MUST include the names of the matching handlers. If the invocation includes a specific action, the localized name of the action SHOULD also be displayed in the prompt.

     b. The most common use cases of the device SHOULD NOT be made more complicated with excessive prompting. For example, the MIDlet installer (JAD handler) SHOULD always be invoked without extra prompting when the user selects a link to a JAD file from the device browser.

- **Justification/Notes:**

  Overriding of handlers is not defined in any previous document, and therefore a potential source for fragmentation.

6.10 Payment API (JSR 229)

JSR 229 was a Component JSR in the Mobile Service Architecture (JSR 248) 1.00, but was removed from the list of mandatory Component JSRs in JSR 248 in a maintenance release. This was done due to issues with Technology Compatibility Kit (TCK) support and licensing.

For the same reasons, JSR 229 is not a Component JSR in MSA 2.

Application developers should note that this creates an exception to the general rule of MSA releases being backward compatible. An application written for MSA 1.00 specification compliant implementations and utilizing JSR 229 may not work on implementations supporting later versions of the MSA specification.

6.11 Advanced Multimedia Supplements (JSR 234)

JSR 234 defines an API for advanced multimedia functionality to supplement MMAPI (JSR 135). The API provides access to advanced media features such as enhanced camera support, 3D audio support, audio radio support, image encoding, and image post-processing capabilities.

6.11.1 Rationale for Inclusion

This API enables the creation of advanced multimedia applications.

6.11.2 Condition for Inclusion

JSR 234 MUST be implemented by compliant implementations. It consists of a single Optional Package, which means that all the classes and interfaces MUST be present in all implementations.

However, some of the underlying capabilities, called Media Capabilities in the JSR 234 specification, are optional or conditionally mandatory, as defined below:

- Implementations MUST support Image Encoding and Image Post-Processing Capabilities.
- If the device has one or more cameras, the Camera Capability MUST be supported. If the device has more than one camera, the requirements stated in JSR 234 specification in “Camera Capability” description SHOULD be fulfilled for all the cameras of the device, not just for the default camera (the one created with `Manager.createPlayer("capture://video");`).
- If the device has an audio radio, the Tuner Capability MUST be supported. If the device audio radio has RDS (Radio Data System for VHF/FM Sound Broadcasting, BS EN 50067:1998) or RBDS (Radio Broadcast Data System, U.S. RBDS Standard – April 1998) support, `RDSControl` MUST also be supported for the audio radio tuner `Player`.
- If the device contains hardware or software support for positional three-dimensional (3D) audio, 3D Audio Capability MUST be supported.

6.11.3 Clarifications

None.

6.12 Mobile Internationalization API (JSR 238)

JSR 238 defines a common API for the internationalization of Java ME applications. The API provides the means to isolate localizable application resources from the program source code, and to access those resources at run time. The API supports accessing the correct resources for a user-selected or device-selected locale. The API also supports recognising cultural conventions in applications; for example, for formatting dates, times, numbers, and currencies, and sorting text strings correctly for the selected locale.

6.12.1 Rationale for Inclusion

This small but beneficial feature allows truly global application development.

6.12.2 Condition for Inclusion

None.

6.12.3 Clarifications

No Additional Clarifications.

6.13 Java Binding for the OpenGL ES API (JSR 239)

JSR 239 provides Java bindings to the OpenGL ES (Embedded Subset) native 3D graphics library.

6.13.1 Rationale for Inclusion

Although JSR 297 provides an API for Mobile 3D Graphics, some developers are more familiar with OpenGL. JSR 239 provides Java bindings to the OpenGL ES (Embedded Subset) 1.0/1.1 native 3D graphics library.

6.13.2 Condition for Inclusion

None.

6.13.3 Clarifications

No Additional Clarifications.

6.14 Mobile Sensor API (JSR 256)

JSR 256 provides a general Sensor API that extends the usability and choice of sensors for Java ME applications. It defines generic sensor functionality optimized for mobile devices.

6.14.1 Rationale for Inclusion

Mobile devices include a number of sensors that produce measurement readings and other dynamically changing status information obtained from the device. Mobile Sensor API is well suited for exposing this dynamic information to applications in a standardized way.

6.14.2 Condition for Inclusion

None.

6.14.3 Clarifications

6.14.3.1 Battery Charge Sensor and Charger State Sensor

Clarification ID: MSA2.256.001

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Sensor API, Version 1.1, Appendix E, Tables E.8 and E.9

- **Requirement Text:**

  A battery charge sensor MUST be supported as specified in Table E.8 of the Mobile Sensor API specification [JSR256]. This means that calling `SensorManager.findSensors("battery_charge", SensorInfo.CONTEXT_TYPE_DEVICE)` MUST return at least one `SensorInfo` object that has a channel whose `ChannelInfo` object contains the information specified in table E.8.

  A charger state sensor MUST be supported as specified in Table E.9 of the Mobile Sensor API specification [JSR256]. This means that calling `SensorManager.findSensors("charger_state", SensorInfo.CONTEXT_TYPE_DEVICE)` MUST return at least one `SensorInfo` object that has a channel whose `ChannelInfo` object contains the information specified in table E.9.

- **Justification/Notes:**

  Applications may need to know the battery status so that they can avoid operations, such as network connections, when the battery level is too low.

6.14.3.2 Network Signal Strength Sensor

Clarification ID: MSA2.256.002

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Sensor API, Version 1.1, Appendix E, Table E.15

- **Requirement Text:**

  A network signal strength sensor MUST be supported as specified in Table E.15 of the Mobile Sensor API specification [JSR256]. This means that calling `SensorManager.findSensors("network_field_intensity", SensorInfo.CONTEXT_TYPE_DEVICE)` MUST return at least one `SensorInfo` object that has a channel whose `ChannelInfo` object contains the information specified in table E.15.

- **Justification/Notes:**

  Applications may need access to network signal strength information.

6.14.3.3 Accelerometer Sensor

Clarification ID: MSA2.256.003

- **Applicable Document, Section, Classes, and Methods:**

  Mobile Sensor API, Version 1.1, Appendix E

- **Requirement Text:**

  If the device supports an accelerometer sensor that is exposed to downloadable applications through a software API, an accelerometer sensor MUST be supported. This means that calling `SensorManager.findSensors("acceleration", SensorInfo.CONTEXT_TYPE_USER)` MUST return at least one `SensorInfo` object that has a channel for each supported acceleration measurement axis (x, y, z). The `ChannelInfo` objects from these channels MUST contain the following information:

Name	Data type	Unit
`axis_x`	`ChannelInfo.TYPE_DOUBLE`	`m/s^2`
`axis_y`	`ChannelInfo.TYPE_DOUBLE`	`m/s^2`
`axis_z`	`ChannelInfo.TYPE_DOUBLE`	`m/s^2`

  The coordinate system MUST be as described in Figure E.1 and related text in appendix E of the Mobile Sensor API specification [JSR256].

- **Justification/Notes:**

  Accelerometer is useful for many application use cases. For example, a game could be controlled by tilting the device, which can be detected using the accelerometer sensor.

6.15 Contactless Communication API (JSR 257)

JSR 257 defines Java ME Optional Packages for contactless communication, including access to Radio-frequancy identification (RFID) and visual tags.

6.15.1 Rationale for Inclusion

Barcode and RFID tag readers are being introduced to a growing number of mobile handsets. Contactless Communication API provides a standardized way to expose this new functionality to applications.

6.15.2 Condition for Inclusion

Contactless Communication API MUST be implemented in devices that expose an RFID reader or a visual tag reader to downloadable 3rd party applications through a software interface

6.15.3 Clarifications

No Additional Clarifications.

6.16 Mobile User Interface Customization API (JSR 258)

The Mobile User Interface Customization API (JSR 258) provides a way to query and modify the user interface customization properties of a mobile device or platform.

6.16.1 Rationale for Inclusion

Application developers may wish to create a consistent look and feel for a number of different applications. In some other cases they might prefer the applications to adopt the look and feel of the native UI system. Mobile User Interface Customization API provides means to implement these key use cases.

6.16.2 Condition for Inclusion

Mobile User Interface Customization API implementation MUST be “fully compliant” as defined in section “Specification Compliance” of the Mobile User Interface Customization API specification [JSR258].

6.16.3 Clarifications

No additional clarifications.

6.17 MIDP 2.1 (JSR 118) and MIDP 3 (JSR 271)

JSR 118 defines the Mobile Information Device Profile version 2.1 (MIDP 2.1). JSR 118 is a mandatory component in MSA 2 Entry Platform implementations.

JSR 271 defines the Mobile Information Device Profile version 3.0 (MIDP 3.0). JSR 271 is a mandatory component in MSA 2 Standard Platform and MSA 2 Advanced Platform implementations.

6.17.1 Rationale for Inclusion

MIDP is a fundamental component of MSA.

6.17.2 Condition for Inclusion

None.

6.17.3 Clarifications

6.17.3.1 Minimum Size of TextBox and TextField UI Elements

Clarification ID: CID.118.1

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, pages 321-342.

  MIDP 3 Specification

  Class `javax.microedition.lcdui.TextBox`
  Class `javax.microedition.lcdui.TextField`

- **Requirement Text:**

  The minimum size (storage capacity) of a `TextField` or a `TextBox` MUST NOT be less than 1000 characters.

- **Justification/Notes:**

  The MIDP specification does not define the minimum capacity of `TextField` and `TextBox` UI elements. Therefore, this requirement is needed to improve the predictability of implementations.

6.17.3.2 Image Object Size

Clarification ID: CID.118.2

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, page 16.

  MIDP 3 Specification

- **Requirement Text:**

  Implementations MUST support at least the creation of `Image` objects (regardless of the format) with sizes equal to (screen width) by (screen height) by (color depth in bits) or the number of bytes specified in the table below, whichever value is greater. Note that the internal representation of an `Image` object SHOULD hold at least 16 bits of color/transparency data per pixel.

	EP	**SP**	**AP**
Minimum Image object size	262144 bits = 32768 B	619520 bits = 77440 B	1382400 bits = 172800 B

  This clarification does not supersede the “Bitmaps Minimums” clarification in the “MIDP-Related JTWI Clarifications” subsection. Implementations MUST satisfy requirements from both clarifications.

- **Justification/Notes:**

  Definition of a minimum supported `Image` object size improves the predictability of implementations.

6.17.3.3 Support for Communication Protocols

Clarification ID: CID.118.3

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, pages 55-60.

  MIDP 3 Specification.

  Package `javax.microedition.io`

- **Requirement Text:**

  Implementations MUST provide support for the following network communication protocols, that is, provide an implementation of the following Java ME interfaces:

  - `javax.microedition.io.SocketConnection`

- `javax.microedition.io.SecureConnection`
- `javax.microedition.io.HttpsConnection`

- **Justification/Notes:**

  The MIDP specification describes the possibility of supporting network communication protocols but leaves their implementation optional. This requirement improves the predictability of implementations.

6.17.3.4 Protocols Supported by PushRegistry

Clarification ID: CID.118.4

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, pages 98-101.

  MIDP 3 Specification.

  Class `javax.microedition.io.PushRegistry`

- **Requirement Text:**

  MIDP `PushRegistry` MUST support the following protocols and services:

  - Bluetooth RFCOMM (Conditional on support for Bluetooth API. For more information, see the “Java APIs for Bluetooth (JSR 82)” section in the “Additional Clarifications” chapter).
  - Bluetooth L2CAP (Conditional on support for Bluetooth API. For more information, see the “Java APIs for Bluetooth (JSR 82)” section in the “Additional Clarifications” chapter).
  - SMS
  - MMS
  - SIP (Conditional on support for SIP API. For more information, see the “SIP API for J2ME (JSR 180)” section in the “Additional Clarifications” chapter).

  The exact requirements for implementing `PushRegistry` support for a particular protocol or service are defined in the corresponding JSR specifications.

- **Justification/Notes:**

  The MIDP specification does not define which protocols and services must be supported by the `PushRegistry`. Mandating `PushRegistry` support for certain protocols and services improves the predictability of implementations.

6.17.3.5 Number of Application-Created Threads per MIDlet

Clarification ID: CID.118.5

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, pages 15-16.

  MIDP 3 Specification.

- **Requirement Text:**

  Implementations MUST allow a MIDlet to create a minimum of 10 simultaneously running threads.

- **Justification/Notes:**

  A need exists to define a minimum number of application-created threads that every implementation must support.

6.17.3.6 MIDlet Suite and LIBlet Download and Install Sizes

Clarification ID: CID.118.6

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, pages 15-16.

  MIDP 2.1 Specification, Chapter 2 “Over The Air User Initiated Provisioning Specification”, pages 19-29.

  MIDP 3 Specification.

- **Requirement Text:**

  Implementations MUST provide the following minimum storage capacity for JAD and JAR files to support OTA download and installation of MIDlet suites and LIBlets (introduced in MIDP 3):

	EP	**SP**	**AP**
JAD size	10 kB	20 kB	20 kB
JAR size	150 kB (300kB recommended)	512 kB (1024 kB recommended)	1024 kB (2048 kB recommended)

  This requirement does not apply if there is not enough room to store a MIDlet suite because the storage space is filled with other downloaded content (including other MIDlet suites).

- **Justification/Notes:**

  For OTA download and installation of MIDlet suites and LIBlets a need exists to define a minimum supported size for JAD and JAR files.

6.17.3.7 Number and Sizes for Attributes in JADs and Manifests

Clarification ID: CID.118.7

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, pages 15-16.

  MIDP 2.1 Specification, Chapter 2 “Over The Air User Initiated Provisioning Specification”, pages 19-29.

  MIDP 3 Specification.

- **Requirement Text:**

  Implementations MUST support at least the minimum number of attributes in both JAD files and manifests as specified in the table below.

	EP	SP	AP
Number of individual attributes in JAD/Manifest	100	512	512

  Implementations MUST support JAD attribute value size of at least 2048 bytes in both JAD files and manifests.

  Implementations MUST support attribute name size of 70 bytes in both JAD files and manifests. The JAR File Specification [JAR] defines the maximum length for an attribute name to be 70 bytes.

- **Justification/Notes:**

  This clarification defines minimum supported numbers and sizes of attributes for both JAD files and manifests, thus improving the predictability of implementations.

6.17.3.8 Number of MIDlets in MIDlet Suite

Clarification ID: CID.118.8

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, pages 15-16.

  MIDP 3 Specification.

- **Requirement Text:**

  Implementations MUST support MIDlet suites containing any number of MIDlets between one and five (inclusive). Implementations MAY support MIDlet suites containing more than five MIDlets.

  This requirement applies to suite installation and MIDlet presentation in the device UI. It does not apply to the ability to run MIDlets simultaneously.

- **Justification/Notes:**

  A need exists to define a minimum supported number of MIDlets in a MIDlet suite (for the purposes of installation, presentation in the UI, and so on). The goal of this clarification is to improve the predictability of implementations.

6.17.3.9 RMS Data Size per MIDlet Suite

Clarification ID: CID.118.9

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, pages 15-16.

  MIDP 3 Specification.

- **Requirement Text:**

  Implementations MUST be able to honor requests by MIDlet suites for an RMS data size of at least the amount specified in the RMS_MINIMUM-row in the table below . This means that if a MIDlet suite requests to reserve RMS_MINIMUM (or less) for its RMS data (using its `MIDlet-Data-Size` attribute), the implementation MUST reserve the requested amount of memory for `RecordStores` created by MIDlets from this suite. This requirement does not apply in cases where insufficient memory is available to honor the request because the device memory is occupied with other data (including `RecordStores` of other MIDlet suites).

	EP	**SP**	**AP**
RMS_MINIMUM	65536 (= 64 kB)	262144 (= 256 kB)	524288 (= 512 kB)

  However, if an implementation honors a request for a certain size of RMS data, this amount of memory MUST be available to MIDlets from the suite during the entire time the suite is installed on the device.

  More precisely, an implementation MUST ensure the following:

  - A MIDlet suite requesting the RMS data size less than or equal to RMS_MINIMUM in the `MIDlet-Data-Size` attribute can be successfully installed and MIDlets from that suite can run.

- If a MIDlet creates a single (empty) `RecordStore`, the `RecordStore.getSizeAvailable()` method returns a value greater than or equal to the value of the `MIDlet-Data-Size` attribute.
- At any time, a MIDlet can store a single byte array of at least `RecordStore.getSizeAvailable()` bytes using the `RecordStore.addRecord()` method. In particular, a single byte array size of the value of the `MIDlet-Data-Size` attribute can be stored in an empty record store.

- **Justification/Notes:**

  This clarification defines a minimum supported RMS data size per MIDlet suite, thus improving the predictability of implementations

6.17.3.10 Number of Independent RecordStores per MIDlet Suite

Clarification ID: CID.118.10

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, pages 15-16.

  MIDP 3 Specification.

- **Requirement Text:**

  Implementations MUST permit a MIDlet suite to create at least 10 independent `RecordStores`.

- **Justification/Notes:**

  This clarification defines a minimum supported number of independent `RecordStores` per MIDlet suite thus improving the predictability of implementations.

6.17.3.11 Behavior of MIDlet.platformRequest()

Clarification ID: CID.118.11, MSA2.118.004

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, pages 443-444.

  MIDP 3 Specification.

  Method `javax.microedition.midlet.platformRequest()`

- **Requirement Text:**

  Method `MIDlet.platformRequest()` MUST support at least the following types of requests:

- URL pointing to a web site (`http://...` , according to RFC 2616 [RFC2616]). In this case, the implementation MUST either launch the inbuilt browser application, bring it to the foreground while keeping the MIDlet running in the background, or wait until the MIDlet exits and then start the browser application. In the former case the `MIDlet.PlatformRequest()` method MUST return false; in the latter case it MUST return true. If the device allows some other browser application to be set as the default browser, that MAY be used instead of the inbuilt one.

- URL pointing to a phone number (`tel:<number>`, according to RFC 3966 [RFC3966]). In this case, the implementation MUST either launch the inbuilt call application and bring it to the foreground while keeping the MIDlet running in the background, or wait until the MIDlet exits and then start the call application. In the former case the `MIDlet.PlatformRequest()` method MUST return false; in the latter case it MUST return true. If the device allows some other call application to be set as the default call application, that MAY be used instead of the inbuilt one.

- URL defining an Internet mail address (`mailto:...`, according to RFC 2368 [RFC2368]). In this case, if the device supports an inbuilt e-mail client application, the implementation MUST either, launch the e-mail client and bring it to the foreground while keeping the MIDlet running in the background, or wait until the MIDlet exits and then start the e-mail client application. When the e-mail client is launched (or brought to the foreground), a message to the designated address MUST be created and the various header fields set as requested in the mailto URL. After creating the message, the user MUST be allowed to edit the message, send it unedited, or choose not to send the message. If the device allows some other e-mail application to be set as the default e-mail client, that MAY be used instead of the inbuilt one.

If a URL passed in to the `platformRequest()` method points to a MIDlet suite residing in the network (either to a JAD or a JAR file), an implementation MUST interpret the method call as a request to install the suite. In this case, the normal MIDlet suite OTA provisioning process MUST be performed, and the user MUST be allowed to control the process (including cancelling the download, the installation, or both). If the MIDlet suite being installed is an update of an installed MIDlet suite, and if some MIDlets from that suite are currently being executed, the platform MUST stop all such MIDlets before performing the update.

- **Justification/Notes:**

  This clarification contains additional requirements for implementations of the `MIDlet.platformRequest()` method. These requirements improve the predictability of the method behavior in different implementations.

6.17.3.12 WAV and JPEG Formats Are Mandatory

Clarification ID: CID.118.12

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification, Section 1.4 “Device Requirements”, page 17.

MIDP 3 Specification.

- **Requirement Text:**

  MSA 2 SP and AP implementations MUST support the wav format (8-bit, 8 KHz, mono linear PCM wav format).

  MSA 2 EP implementations MUST support the wav format (8-bit, 8 KHz, mono linear PCM wav format) if the device has a native player for the format.

  Implementations MUST support sampled audio.

  Implementations MUST support the JPEG format [JPEG].

- **Justification/Notes:**

  In the MIDP Specification, the wav format (8-bit, 8 KHz, mono linear PCM wav format) is mandatory *only* in a case where the device supports sampled audio. This clarification requires unconditional support for the wav format, sampled audio and JPEG format thus improving the predictability of implementations.

6.17.3.13 Support for Alpha Blending

Clarification ID: MSA2.118.008

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification.

  MIDP 3 Specification.

  Class `javax.microedition.lcdui.Display`
  Method `numAlphaLevels()`

  Class `javax.microedition.lcdui.Graphics`
  Method `drawImage()`
  Method `drawRegion()`
  Method `drawRGB()`

  Class `javax.microedition.lcdui.Image`
  Method `createImage(byte[] imageData, int imageOffset, int imageLength)`
  Method `createImage(Image source)`
  Method `createImage(Image image, int x, int y, int width, int height, int transform)`
  Method `createImage(InputStream stream)`
  Method `createRGBImage()`

- **Requirement Text:**

  At least 16 levels for alpha blending MUST be supported by the implementation.

- **Justification/Notes:**

  MIDP 3 specification requires 16 levels to be supported for alpha blending. This MSA 2 specification requirement sets the same requirement for MIDP 2.1 devices, making MSA 2 compliant implementations more consistent with each other.

6.17.3.14 MIDlet Operation in Clamshell Devices

Clarification ID: MSA2.118.018

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification.

  MIDP 3 Specification.

  Class `javax.microedition.midlet.MIDlet`
  Method `pauseApp()`
  Method `destroyApp(boolean unconditional)`

- **Requirement Text:**

  If a clamshell device generally allows downloadable applications to execute when the flip is closed, MIDlets MUST be allowed the same.

- **Justification/Notes:**

  Several application use cases require MIDlets to continue executing when the flip is closed. For example, a stock tracker application should continue running even when the flip is closed. This clarification reduces the number of device implementations that unnecessarily terminate or pause MIDlets when the flip is closed.

6.17.3.15 HTTPS Certificate Requirements

Clarification ID: MSA2.118.010

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 Specification.

  MIDP 3 Specification.

  Package `javax.microedition.io`
  Interface `HttpsConnection`

- **Requirement Text:**

  Certificates available in public certificate stores of the inbuilt browser for use with HTTPS connections MUST also be available for HTTPS connections from MIDlets. Certificates in non-public certificate stores MAY not be available for MIDlets.

- **Justification/Notes:**

  This requirement improves the consistency of the HTTPS functionality provided by the inbuilt browser and the Java environment.

6.17.3.16 User-Agent Header and the Discovery Agent

Clarification ID: MSA2.118.013

- **Applicable Document, Section, Classes, and Methods:**

  MIDP 2.1 and MIDP 3 Specifications, Sections “MIDlet suite discovery” and “Device Identification and Request Headers”

- **Requirement Text:**

  Implementations MUST ensure that the Discovery Agent(s) prebuilt into the device include the product tokens for device manufacturer, device model number and device software version in the HTTP User-Agent request header.

- **Justification/Notes:**

  The MIDP specifications define that the Discovery Agent MUST include the HTTP User-Agent header, and that it MUST include information about the device manufacturer and model number when requesting the Application Descriptor from the Discovery Agent. This requirement ensures that the information relayed in the User-Agent header contains the necessary tokens to properly identify the device.

6.18 Mobile Broadcast Service API (JSR 272)

JSR 272 provides functionality to handle broadcast content, e.g. to view digital television and to utilize its rich features and services.

6.18.1 Rationale for Inclusion

Digital television receiver functionality is being introduced to a growing number of mobile handsets. Mobile Broadcast Service API provides a standardized way to expose this new functionality to applications.

6.18.2 Condition for Inclusion

Mobile Broadcast Service API MUST be implemented in devices that expose a digital broadcast television tuner to downloadable 3rd party applications through a software interface.

6.18.3 Clarifications

No Additional Clarifications.

6.19 XML API for Java ME (JSR 280)

JSR 280 defines a common general purpose XML API for mobile devices.

6.19.1 Rationale for Inclusion

This API replaces the XML parsing in JSR 172 and provides a common XML parsing API for other JSRs.

6.19.2 Condition for Inclusion

None.

6.19.3 Clarifications

No Additional Clarifications.

6.20 IMS Services API (JSR 281)

JSR 281 provides a high-level API to access IP Multimedia Subsystem (IMS) services. This API hides IMS technology details and exposes service-level support to enable easy development of IMS applications.

6.20.1 Rationale for Inclusion

IMS services are being introduced to a number of mobile handsets and mobile networks. IMS Services API provides a standardized way to expose this functionality to applications.

6.20.2 Condition for Inclusion

IMS Services API MUST be implemented in devices that expose 3GPP TS 24.229 (IMS call control protocol based on SIP and SDP; Stage 3) functionality to downloadable 3rd party applications through a software interface.

6.20.3 Clarifications

No Additional Clarifications.

6.21 Scalable 2D Vector Graphics API 2.0 for Java ME (JSR 287)

JSR 287 defines an API for rendering scalable 2D vector graphics, including image files in W3C Scalable Vector Graphics (SVG) format. It also defines a rich subset of the uDOM API for user interaction and dynamic manipulation of the SVG content.

6.21.1 Rationale for Inclusion

Scalable vector graphics makes it possible to create, manipulate (zoom and resize), and render graphics content.

6.21.2 Condition for Inclusion

None.

6.21.3 Clarifications

None.

6.22 Java Language & XML UI Markup Integration (JSR 290)

JSR 290 enables creation of Java ME applications which combine Web UI markup technologies with Java code.

6.22.1 Rationale for Inclusion

The API will enable the creation of Java ME applications which combine the ease of authoring and graphical richness of Web UI technologies (such as XHTML Basic and SVG Tiny) with the power, flexibility, and breadth of the Java ME platform.

6.22.2 Condition for Inclusion

None.

6.22.3 Clarifications

No Additional Clarifications.

6.23 Location API 2.0 (JSR 293)

JSR 293 defines an API that enables developers to write mobile location-based applications for devices with limited resources. It is a compact and generic API that produces information on the present physical location of the device.

6.23.1 Rationale for Inclusion

An increasing number of mobile phones have the capability of determining their geographical location. This API enables developers to create new types of location-based applications and to enhance the usability of existing applications with location awareness.

6.23.2 Condition for Inclusion

JSR 293 MUST be implemented if the target device meets at least one of the following conditions:

- The device has a GPS receiver that is able to deliver the geographical coordinates within the device
- The device supports a location method that is capable of delivering the geographical coordinates and is used to deliver the coordinates to downloadable applications (in Java ME or other runtime platforms)
- The device supports an accessory device that can be used to obtain geographical coordinates, and which is used to deliver the location to downloadable applications (in Java ME or other runtime platforms)

6.23.3 Clarifications

No Additional Clarifications.

6.24 Mobile 3D Graphics API 2.0 (JSR 297)

JSR 297 specifies a lightweight, interactive 3D graphics API. The API is flexible enough for a wide range of applications, such as games, animated messages, screen savers, custom user interfaces, and product visualization.

6.24.1 Rationale for Inclusion

3D Graphics are very important for games, rich graphical user interfaces, and other graphics intensive applications.

6.24.2 Condition for Inclusion

Mobile 3D Graphics API 2.0 includes a Core Block that MUST be present in all devices implementing the API. In addition:

- Mobile 3D Graphics API 2.0 implementations MUST support the Advanced Block in devices that expose OpenGL ES 2.0 functionality to downloadable 3rd party applications through a software interface.

Advanced Block is RECOMMENDED to be implemented in all MSA 2 Advanced Platform implementations.

6.24.3 Clarifications

6.24.3.1 Security When Loading M3G Content over Network Connections

Clarification ID: CID.184.1

- **Applicable Document, Section, Classes, and Methods:**

  Mobile 3D Graphics API.

  Class `javax.microedition.m3g.Loader`
  Method `public static Object3D[] load(java.lang.String name)`
  Method `public static Object3D[] load(byte[] data, int offset)`

- **Requirement Text:**

  If the `load` method is used for loading content from a resource that is accessed by using a networking protocol such as HTTP, this method MUST perform the same security checks as the corresponding API for using the networking protocol directly (when using, for example, `javax.microedition.io.HttpConnection`).

This applies when the `name` parameter directly contains an URI requiring retrieval over a networking protocol, and when a M3G file is loaded and the contents of the file references resources using such URIs.

- **Justification/Notes:**

  JSR 297 leaves open the security aspect of using a networking protocol. This requirement makes it consistent with other APIs that are part of the MSA.

6.24.3.2 Relation to the MIDlet Paused State

Clarification ID: CID.184.2

- **Applicable Document, Section, Classes, and Methods:**

  Mobile 3D Graphics API. This clarification applies to implementations of the JSR 297 API and the MIDlet Paused state in MIDP 2.1.

- **Requirement Text:**

  The implementation of the JSR 297 API MUST NOT automatically release any resources or take any other action as a result of the MIDlet moving to the Paused state.

  However, according to the semantics of the Paused states, applications SHOULD release resources, including any resources held using the JSR 297 API.

- **Justification/Notes:**

  Although the MIDP 2.1 specification specifies the semantics of the Paused state, some confusion exists regarding its relation with some APIs. This clarification specifies that the Paused state does not have any direct relation in the implementation with the JSR 297 API.

6.25 Requirements Inherited from JTWI 1.0 (JSR 185)

The MSA specification is based on the Java Technology for the Wireless Industry Specification, version 1.0 (JTWI Specification, JSR 185).

To ensure maximum backward compatibility, the JTWI requirements are included in the MSA Specification, and a compliant implementation MUST provide support for those requirements.

6.25.1 Omitted Clarifications

This section covers only a subset of the original JTWI clarifications. Those clarifications that are explicitly overridden, superseded, revised, or otherwise rendered unnecessary by MSA are omitted. A summary of the omitted JTWI clarifications is at the end of this section.

6.25.2 Configuration-Related JTWI Clarifications

A compliant implementation MUST support the Configuration-related (CLDC/CDC) clarifications defined in this subsection.

6.25.2.1 Minimum Clock Resolution

Clarification ID: CID.185.1

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 3.3

  CLDC 1.1.1 Specification, CDC 1.1.2 Specification
  Class `java.lang.System`:
  `public long currentTimeMillis()`

- **Requirement Text:**

  Implementations of the method `java.lang.System.currentTimeMillis()` MUST record the elapsed time in increments not to exceed 40 milliseconds. Various factors, such as garbage collection, affect the ability to achieve this requirement. Because of this, at least 80% of test attempts MUST meet the time elapsed requirement to achieve acceptable conformance.

- **Justification/Notes:**

  Implementations MAY support a smaller increment.

6.25.2.2 Custom Time Zone IDs

Clarification ID: CID.185.2

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 3.4

  CLDC 1.1.1 Specification, CDC 1.1.2 Specification
  Class `java.util.TimeZone:`
  `public String getID()`
  `public static TimeZone getTimeZone(String ID)`

- **Requirement Text:**

  The implementation MUST permit the use of custom time zones adhere to the following time zone format:

  - General time zone: For time zones representing a GMT offset value, the following syntax is used:

    CustomID:

    `GMT` *Sign Hours* : *Minutes*

    `GMT` Sign Hours Minutes

    `GMT` Sign Hours

    Sign: one of:

    + -

    Hours:

    Digit

    Digit Digit

    Minutes:

    Digit Digit

    Digit: one of:

    0 1 2 3 4 5 6 7 8 9

    Hours MUST be between 0 and 23. *Minutes* MUST be between 00 and 59. For example, GMT+10 and GMT+0010 mean ten hours and ten minutes ahead of GMT, respectively. The format is locale independent, and *digits* MUST be taken from the Basic Latin block of the Unicode standard. No daylight saving time transition

schedule can be specified with a custom time zone ID. If the specified string does not match the syntax, GMT is used.

- When creating a `TimeZone`, the specified custom time zone ID is normalized in the following syntax:

  NormalizedCustomID:

  GMT *Sign TwoDigitHours* : *Minutes*

  Sign: one of

  + -

  TwoDigitHours:

  Digit Digit

  Minutes:

  Digit Digit

  Digit: one of

  0 1 2 3 4 5 6 7 8 9

For example, `TimeZone.getTimeZone("GMT-8").getID()` returns `GMT-08:00`.

- **Justification/Notes:**

  Mandating support for the custom time zone format provides consistent time zone ID support across implementations.

6.25.2.3 Names for Encodings

Clarification ID: CID.185.3

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 3.5

  CLDC 1.1.1 Specification, Section 6.2.11, CDC 1.1.2 Specification
  Class `java.io.InputStreamReader`
  Class `java.io.OutputStreamWriter`
  Class `java.lang.String`:

```
public String(byte[] bytes, int off, int len, String enc)
public String(byte[] bytes, String enc)
public byte[] getBytes(String enc)
```

- **Requirement Text:**

  Implementations MUST support at least the preferred MIME name as defined by IANA (`http://www.iana.org/assignments/character-sets`) for the supported character encodings. For example, for ISO 646, the name US-ASCII MUST be supported. If no preferred name has been defined, then the registered name MUST be used, for example, UTF-16.

- **Justification/Notes:**

  Application developers are encouraged to use the preferred names to ensure portability. Although devices are permitted to use the other names, this is intended only for backward compatibility.

6.25.2.4 Character Properties

Clarification ID: CID.185.4

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 3.6

  CLDC 1.1.1 Specification, Section 6.2.1, CDC 1.1.2 Specification

```
Class java.lang.String:
     public int compareTo(String s)
     public boolean regionMatches(boolean ignoreCase, int toffset,
                                  String other, int ooffset,
                                  int len)
Class java.lang.Character:
     public boolean equals(Object o)
     public static boolean isLowerCase(char ch)
     public static boolean isUpperCase(char ch)
     public static boolean isDigit(char ch)
     public static char toLowerCase(char ch)
     public static char toUpperCase(char ch)
     public static int digit(char ch, int radix)
```

- **Requirement Text:**

  Implementations MUST provide support for character properties and case conversions for characters in the Basic Latin and Latin-1 Supplement blocks of Unicode 3.0. Other Unicode character blocks MAY be supported as necessary.

- **Justification/Notes:**

  The CLDC 1.0 specification permits more relaxed case conversion. However, applications which target many international markets will be less satisfactory for their users if case conversion is incomplete.

6.25.3 MIDP-Related JTWI Clarifications

The MIDP-related clarifications defined in this subsection MUST be supported.

Note: This subsection provides only a subset of the original JTWI clarifications. Clarifications that are explicitly overridden, replaced or moved to other specifications are omitted. A summary of the omitted clarifications is at the end of this section.

6.25.3.1 Timer Resolution

Clarification ID: CID.185.5

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 4.5

  MIDP 2.1 Specification, Chapter 6 “Package java.util”, pages 48-52.
  MIDP 3.0 Specification
  Class `java.util.Timer`:

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

- **Requirement Text:**

  An implementation MUST permit an application to specify the values for the `firstTime`, `delay`, and `period` parameters of `java.util.timer.schedule()` methods, with a distinguishable resolution of no more than 40 ms. Various factors, such as garbage collection, affect the ability to achieve this requirement. Because of this, at least 80% of test attempts MUST meet the schedule resolution requirement to achieve acceptable conformance.

- **Justification/Notes:**

  Predictability is crucial to application interoperability. This mandate is provided also to set expectations for application developers.

6.25.3.2 Minimum Number of Timers

Clarification ID: CID.185.6

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 4.6

  MIDP 2.1 Specification, Chapter 6 “Package java.util”, pages 48-52.

MIDP 3.0 Specification
Class `java.util.Timer`:

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

- **Requirement Text:**

  Implementations MUST allow a MIDlet suite to create a minimum of 5 simultaneously running Timers. This requirement is independent of the minimum application thread count requirements.

- **Justification/Notes:**

  Predictability is crucial to application interoperability. This mandate is also provided to set expectations for application developers. This requirement is not intended to require implementations to guarantee at all times that 5 timers be possible, but to require that implementations not artificially limit timer creation to less than 5 timers. Application developers MUST still manage resource usage within the physical constraints of the device.

6.25.3.3 Bitmap Minimums

Clarification ID: CID.185.7

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 4.7

  MIDP 2.1 Specification, Chapter 8 “Package javax.microedition.lcdui”, , pages 271-281.
  MIDP 3.0 Specification
  Class `javax.microedition.lcdui.Image`:

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

- **Requirement Text:**

  Implementations MUST support the loading of PNG [PNG] images with pixel color depths of 1, 2, 4, 8, 16, 24, and 32 bits per pixel, per the PNG image format specification. For each of these color depths, as well as for JPEG image formats, implementations MUST support images of up to 32768 total pixels.

- **Justification/Notes:**

  Devices MUST be able to process color images even if they do not actually have a color display.

6.25.3.4 TextField, TextBox and Phone Book Coupling

Clarification ID: CID.185.8

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 4.8

  MIDP 2.1 Specification, Chapter 8 “Package javax.microedition.lcdui”, pages 321-342.
  MIDP 3.0 Specification
  Class `javax.microedition.lcdui.TextField`
  Class `javax.microedition.lcdui.TextBox`

- **Requirement Text:**

  Implementations MUST provide a mechanism for selecting a phone number from the device phone book when the user is editing a `TextBox` or `TextField`, and the constraint of the `TextBox` or `TextField` is `TextField.PHONENUMBER`, This requirement is voided if the phone book is not accessible.

- **Justification/Notes:**

  Applications that use wireless messaging are more usable if the user is provided with a simple means of copying data from the phone book. Other applications may also benefit from this mechanism. Note that at no time SHOULD the MIDlet be given direct access to the contents of the device phone book itself. This is intended as a convenience for the user to select a number from the stored phone book and insert it into the text field.

6.25.3.5 PushRegistry Alarm Events

Clarification ID: CID.185.9

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 4.11

  MIDP 2.1 Specification, Chapter 7 “Package javax.microedition.io”, pages 98-108.
  MIDP 3.0 Specification
  Class `javax.microedition.io.PushRegistry`

- **Requirement Text:**

  Implementations MUST support alarm-based push registry entries. If no other security mechanism is present, the `PushRegistry Alarm` function MUST NOT be allowed without explicit user permission.

- **Justification/Notes:**

  This provides a practical means for alarm and calendar-based applications.

6.25.4 WMA-Related JTWI Clarifications

A compliant implementation MUST support the WMA-related clarifications defined in this subsection.

6.25.4.1 Support for SMS in GSM Devices

Clarification ID: CID.185.10

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 5.2

  WMA 2.0 Specification, Appendix A

- **Requirement Text:**

  GSM/WCDMA (UMTS) phones MUST support the GSM SMS service by using this API as specified in Appendix A of the Wireless Messaging API (WMA) Specification.

- **Justification/Notes:**

  Device-to-device communication is a critical element of a broad range of mobile applications. The JTWI Specification requires this service to enable these new application classes.

6.25.4.2 Cell Broadcast in GSM Devices

Clarification ID: CID.185.11

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 5.3

  WMA 2.0 Specification, Appendix B

- **Requirement Text:**

  If the implementation supports access to GSM Cell Broadcast via Java APIs, it MUST follow the specification found in Appendix B of the Wireless Messaging API (WMA) Specification.

- **Justification/Notes:**

  Although not every device or network supports Cell Broadcast, uniform access on the devices that do support it is important to improving application portability.

6.25.4.3 SMS Push

Clarification ID: CID.185.12

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 5.4

  WMA 2.0 Specification, Appendix A
  MIDP 2.1 Specification, Chapter 7 “Package javax.microedition.io”, pages 98-108.
  MIDP 3.0 Specification
  Class `javax.microedition.io.PushRegistry`

- **Requirement Text:**

  GSM/WCDMA (UMTS) phones MUST support MIDP 2.1 Push handling for the SMS protocol as specified in Appendix E of the Wireless Messaging API (WMA) Specification. If no other security mechanism is present, the PushRegistry SMS Push function MUST NOT be allowed without explicit user permission.

- **Justification/Notes:**

  SMS Push provides a mechanism for sophisticated server-driven applications. Role-playing and other games, as well as various types of notification, and event-driven applications, such as travel reservation systems, can benefit from this capability. However, there are security ramifications which devices MUST consider when implementing this service.

6.25.5 MMAPI-Related JTWI Clarifications

A compliant implementation MUST support the MMAPI-related clarifications defined in this subsection.

Note: This subsection provides only a subset of the original JTWI clarifications. Clarifications that are explicitly overridden or revised by MSA are omitted. A summary of the omitted clarifications is at the end of this section.

6.25.5.1 HTTP 1.1 Protocol

Clarification ID: CID.185.13

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 6.2

  MMAPI 1.2 Specification, Section “Overview of MMAPI”
  Class `javax.microedition.media.Manager:`
  ```
  public static String getSupportedProtocols()
  public static Player createPlayer()
  public static String getSupportedContentTypes()
  ```

- **Requirement Text:**

  HTTP 1.1 MUST be supported for media file download for all supported media formats.

- **Justification/Notes:**

  MMAPI does not specify any mandatory protocols. It specifically states that protocols MUST be defined in profiles. In keeping with the general-purpose nature of MMAPI, it leaves the supported set of protocols and content formats up to the implementation. It also does not make any requirements on which content types should work over which protocol.

6.25.5.2 JPEG Encoding in Video Snapshots

Clarification ID: CID.185.14

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 6.5

  MMAPI 1.2 Specification, Section javax.microedition.media

  Class `javax.microedition.media.Player:`
  ```
  public Control getControl(String controlType)
  public Control[] getControls()
  ```

- **Requirement Text:**

  A compliant implementation that supports the video feature set and video image capture MUST support JPEG encoding in Video Snapshots, with the same detailed defitions regarding the JPEG image format as specified in the JSR 118 MIDP 2.1 specification for JPEG usage in LCDUI.

- **Justification/Notes:**

  The PNG image format is not always suitable for photographic applications; JPEG is generally more compact and appropriate when storing photographic images.

6.25.5.3 Tone Sequence File Format

Clarification ID: CID.185.15

- **Applicable Document, Section, Classes, and Methods:**

  JTWI 1.0 Specification, Section 6.6

  MMAPI 1.2 Specification, Section javax.microedition.media

  Class `javax.microedition.media.Manager:`

```
public static String[] getSupportedContentTypes(
                                        String protocol)
public static Player createPlayer(DataSource source)
public static Player createPlayer(InputStream stream,
                                        String type)
public static Player createPlayer(String locator)
```

- **Requirement Text:**

  Tone sequence file format MUST be supported.

- **Justification/Notes:**

  Tone sequences provide an additional simple format for supporting the audio needs of many types of games and other applications.

6.25.6 Summary of Omitted JTWI Clarifications

A number of JTWI clarifications are omitted from this specification because those clarifications were explicitly overridden, superseded, replaced, or otherwise rendered unnecessary by the MSA Specification. The following list summarizes the omitted JTWI clarifications:

- JTWI 1.0 Specification, Section 3.2,"Minimum Application Thread Count"
- JTWI 1.0 Specification, Section 3.7, "Unicode Version"
- JTWI 1.0 Specification, Section 4.2, "Record Store Minimum"
- JTWI 1.0 Specification, Section 4.3, "HTTP Support for Media Content"
- JTWI 1.0 Specification, Section 4.4, "JPEG for Image Objects"
- JTWI 1.0 Specification, Section 4.9, "Supported Characters in TextField and TextBox"
- JTWI 1.0 Specification, Section 4.10, "Supported Characters in EMAILADDR and URL Fields"
- JTWI 1.0 Specification, Section 4.12, "Identification of JTWI via System Property"

- JTWI 1.0 Specification, Section 6.3, “MIDI Feature Set”
- JTWI 1.0 Specification, Section 6.4, “Controls for MIDI Feature Set”
- JTWI 1.0 Specification, Section 7, “Security Policy for GSM/UMTS Compliant Devices”

7. Additional Requirements (normative)

The following sections contain additional requirements for MSA 2 compliancy.

7.1 Hardware Requirements

This chapter contains hardware requirements and recommendations for compliant implementations. These requirements and recommendations are provided for implementers of this specification, and to set application developers' expectations regarding minimum values of certain hardware parameters across all compliant implementations.

7.1.1 Screen Size

Clarification ID: CID.248.1

Minimum screen resolution available to a Java application on a compliant implementation SHOULD be no less than the number of pixels listed in the following table.

	EP	SP	AP
Requirement	128x128	176x220 or 220x176	240x320 or 320x240

The screen resolution is returned by `Canvas.getHeight()` and `Canvas.getWidth()` in full-screen mode.

7.1.2 Color Depth

Clarification ID: CID.248.2

Minimum color depth (as returned by `Display.numColors()`) available to a Java application on a compliant implementation SHOULD be no less than the number of colors defined in the following table.

	EP	SP	AP
Requirement	4096 (12 bits)	65536 (16 bits)	262144 (18 bits)

7.1.3 Java Heap Size Available to MIDlet

Clarification ID: CID.248.3

Under normal conditions (see the definition below), a compliant implementation MUST ensure that Java heap size available to a MIDlet is at least as defined on the "Requirement" row of the table below. Further, under normal conditions, a compliant implementation

SHOULD ensure that Java heap size available to a MIDlet is at least as defined on the "Recommendation" row of the table below.

	EP	**SP**	**AP**
Requirement	512 kB	1024 kB	2048 kB
Recommendation	600 kB	2048 kB	4096 kB
N	7 (512 kB/64kB-1)	15 (1024kB/64kB-1)	31 (2048kB/64kB-1)

Normal conditions are defined as follows:

- A new, "out of the box" device is switched on, and a MIDlet suite is installed. There is only one MIDlet in a MIDlet suite.
- The device is switched off and then switched back on. The MIDlet is started.

One reference method to test the Java heap size requirement is defined as follows:

- A MIDlet started under normal conditions is able to successfully allocate N byte arrays, of size 64 kB each, where N is as defined in the above table. (Note: The number of byte arrays is intentionally one less than the number of 64 kB arrays that would fit into the required heap size. It is assumed that the remaining 64 kB are used to store the MIDlet's class files in their runtime form, and to store the MIDlet's internal runtime data structures.)

  Here is sample code that illustrates this test method:

```
// N = 7, 15 or 31 depending on the platform
byte[][] container = new byte[N][];

for (int i = 0; i < N; i++) {
    byte[] barray = new byte[65536];
    container[i] = barray;
}
```

7.2 Security Requirements

This section contains API permission mappings to Function Groups along with related security requirements for all MSA 2 component JSRs. The section also defines a recommended security policy (Table 1).

This section complements and partly supersedes the Recommended Security Policy in MIDP 2.1 for MSA 2 Entry Platform implementations and the Security Policy in MIDP 3.0 for MSA 2 Standard and Advanced Platform implementations.

For MSA 2 EP, the Recommended Security Policy of MIDP 2.1 MUST be supported. This means that compliant implementations MUST support all of the MIDP 2.1 specification requirements that are labelled MUST. Any MIDP 2.1 specification items that are labelled MAY, RECOMMENDED or SHOULD are to be interpreted as MAY, RECOMMENDED or SHOULD requirements for compliant implementations as well.

For MSA 2 SP and AP, the MIDP 3.0 Security Policy MUST be supported. Correspondingly, this means that compliant implementations MUST support all of the MIDP 3.0 specification requirements that are labelled MUST. Any MIDP 3.0 specification items that are labelled MAY, RECOMMENDED or SHOULD are to be interpreted as MAY, RECOMMENDED or SHOULD requirements for compliant implementations as well.

7.2.1 General

Compliant MSA 2 EP implementations that implement the recommended security policy (Table 1) MUST follow the security framework specified in the MIDP 2.1 Specification. Compliant MSA 2 SP and AP implementations that implement the recommended security policy (Table 1) MUST follow the security framework specified in the MIDP 3.0 Specification. Additionally, devices MUST support the Identified Third Party Protection Domain and, to accomplish this, MUST follow the PKI-based authentication scheme as defined in the MIDP 2.1 specification (MSA 2 EP) or in the MIDP 3.0 specification (MSA 2 SP and AP).

7.2.2 Permissions for Downloaded MIDlet Suites

Certain APIs are considered security sensitive because their abuse by an application can be damaging (financially or otherwise) to the user. Examples of such APIs include APIs that cause consumption of network resources resulting in charges to the user and APIs that allow access to user's private data, thus creating privacy concerns. In order to protect the user and the integrity of the network security sensitive APIs are controlled by user permissions. To make this control more manageable these permissions are grouped logically together by functional area.

The following sections define a recommended security policy (Table 1) comprising of the Function Group definitions, the associated API mappings and the default and other user settings.

7.2.2.1 Mapping Permissions to Function Groups in Protected Domains

A device with small display may not be able to present all permissions on an API level to the user in a single configuration settings menu in a user-friendly manner. Therefore, the device is NOT REQUIRED to present all individual permissions for user confirmation. Rather, a certain higher-level action triggered by the protected function SHOULD be brought to the user for acceptance. The high-level functions presented to the user essentially capture and reflect the actions and consequences of the underlying individual permissions. These so-called function groups are as follows:

Network cost-related groups:

- **Phone Call** – the group represents permissions to any function that results in a voice call.
- **Call Control** – the group represents permissions to any function that results call setup or teardown of a restricted network connection.
- **Net Access** – the group represents permissions to any function that results in an active network data connection (for example, GSM, GPRS, UMTS, EDGE, WiMax, etc.); such functions MUST be mapped to this group.
- **Low Level Net Access** – the group represents permissions to any function that results in an active low level network data connection (for example, Sockets, etc.); such functions MUST be mapped to this group.
- **Messaging-Receive** – the group represents permissions to any function that allows receiving messages (for example, SMS, MMS, etc.).
- **Messaging-Send** – the group represents permissions to any function that allows sending messages (for example, SMS, MMS, etc.).
- **Restricted Messaging-Receive** – the group represents permissions to any function that allows receiving messages from a restricted messaging service (for example, Cell Broadcast, etc.).
- **Restricted Messaging-Send** – the group represents permissions to any function that allows sending messages to a restricted messaging service (for example, Cell Broadcast, etc.).
- **Application Auto Invocation** – the group represents permissions to any function that allows a MIDlet suite to be invoked automatically (for example, push, timed MIDlets, etc.).
- **Local Connectivity** – the group represents permissions to any function that activates a local port for further connection (for example, COMM port, IrDa, Bluetooth, etc.).
- **Authentication** – the group represents permissions to any function that gives a MIDlet suite access to authentication functionality.

User privacy-related groups:

- **Multimedia Recording** – the group represents permissions to any function that gives a MIDlet suite the ability to do any kind of multimedia recording (for example, capture still images, or to record video or audio clips).
- **Read User Data Access** – the group represents permissions to any function that gives a MIDlet suite the ability to read a user's phone book, or any other data in a file or directory.
- **Write User Data Access** – the group represents permissions to any function that gives a MIDlet suite the ability to add or modify a user's phone book or any other data in a file or directory.
- **Smart Card Communication** – the group represents permissions to any function that gives a MIDlet suite the ability to communicate with the smart card.
- **Location** – the group represents permissions to any function that gives a MIDlet suite access to Location information.
- **Landmark** – the group represents permissions to any function that gives a MIDlet suite access to Landmark information.
- **Private Sensor Access** – the group represents permission to any function that gives the MIDlet suite access to data from private sensors. An example of a private sensor could be a heart rate monitor sensor.
- **Mobile Broadcast Service** – the group represents permission to any function that gives the MIDlet suite access to reading and modifying user data related to the Mobile Broadcast Service API (JSR 272). This includes managing scheduled recordings and obtaining information about the user’s purchased subscriptions and other information about the user’s viewing habits.

Resource protection-related groups:

- **NFC Write Access** – the group represents permissions to any function that gives a MIDlet suite the ability to write a Near Field Communication (NFC) compliant tag.
- **UI Customization** – the group represents permissions to customize the device UI. This includes permissions to remove and modify user customization elements, such as UI themes that the user may have paid for.

Whenever new features are added to any of the component JSRs of the MSA Specification, they SHOULD be assigned to the appropriate function group. In addition, APIs that are specified elsewhere (that is, in other JSRs), but rely on the MIDP security framework, SHOULD also be assigned to an appropriate function group. If none of the function groups defined in this section is able to capture the new feature and reflect it to the user adequately, a new function group MUST be defined in this document by requesting an update to this document from MSA.

Tables 1 and 2 contain information partly already defined in the MIDP 3.0 Security Policy. These tables are included here so that the same definitions will be applied to MIDP 2.1 based MSA 2 EP implementations.

Table 1 defines the policy that is RECOMMENDED by the MSA specification. This policy relies on the security framework defined in MIDP 3.0. The table specifies the available permission settings for each function group defined. Settings that are effective at the time the MIDlet suite is invoked for the first time, and remain effective until the user changes them in the MIDlet suite's configuration menu, are called "default settings." Settings available to the user in the configuration menu, to which the user can change from a default setting, are called "other settings." Together, default and other settings form a pool of available configuration settings for the MIDlet suite. Default and other settings are presented for each function group and both Third Party Protection Domains. The naming of the function groups is implementation specific, but MUST follow the guidelines of the function group names defined in this document, as well as the definitions of these groups.

Tables 2 through 17 present individual permissions defined in the MSA 2 Component JSRs and map to the function groups specified in this section. An individual permission MUST occur in only one function group.

A compliant implementation SHOULD adhere to Table 1 in this specification. A compliant implementation MUST adhere to tables 2 through 17 and all associated text accompanying these tables.

It is RECOMMENDED that MIDlet suites in the Manufacturer and Operator Protection Domains adhere to the permission guidelines provided in the tables, and that they present appropriate prompts to the user for the functions identified as security protected.

Table 1: Function Groups and User Settings for Third-Party Protection Domains.

Function Group	**Identified Third Party Protection Domain**		**Unidentified Third Party Protection Domain**	
Phone Call	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Call Control	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Net Access	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
Low Level Net Access	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
Messaging-Send	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Messaging-Receive	default setting	Blanket	default setting	Blanket
	other settings	Oneshot, Session, No	other settings	Oneshot, Session, No
Restricted Messaging-Send	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Restricted Messaging-Receive	default setting	Blanket	default setting	Blanket
	other settings	Oneshot, Session, No	other settings	Oneshot, Session, No
Application Auto	default setting	Oneshot	default setting	Oneshot

Function Group	Identified Third Party Protection Domain		Unidentified Third Party Protection Domain	
Invocation	other settings	Blanket, Session, No	other settings	Session, No
Local Connectivity	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Blanket, Session, No
Multimedia Recording	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot No	other settings	Session, No
Read User Data Access	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Write User Data Access	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Location	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
Landmark	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
Smart Card Communication	default setting	Session	default setting	No
	other settings	Blanket, Oneshot, No	other settings	No
Authentication	default setting	Session	default setting	No
	other settings	Blanket, Oneshot, No	other settings	No
Private Sensor Access	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
NFC Write Access	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
UI Customization	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No
Mobile Broadcast Service	default setting	Session	default setting	Oneshot
	other settings	Blanket, Oneshot, No	other settings	Session, No

Mapping MIDP 2.1 (JSR 118) and MIDP 3.0 (JSR 271) Permissions to Function Groups

Table 2: Assigning Permissions Specified in MIDP 2.1 (JSR 118) and MIDP 3.0 to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Protocol	Function group
`javax.microedition.io.Connector.http`	`javax.microedition.io.HttpProtocolPermission( “http://*”)`	HTTP	Net Access

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Protocol	Function group
javax.microedition.io.Connector.https	javax.microedition.io.HttpsProtocolPermission(“https://*”)	HTTPs	Net Access
javax.microedition.io.Connector.datagram	javax.microedition.io.DatagramProtocolPermission(“datagram://*”)	Datagram	Low Level Net Access
javax.microedition.io.Connector.datagramreceiver	javax.microedition.io.DatagramProtocolPermission(“datagram://”)	Datagram server (without host)	Low Level Net Access
javax.microedition.io.Connector.socket	javax.microedition.io.SocketProtocolPermission(“socket://*”)	Socket	Low Level Net Access
javax.microedition.io.Connector.serversocket	javax.microedition.io.SocketProtocolPermission(“socket://”)	Server socket (without host)	Low Level Net Access
javax.microedition.io.Connector.ssl	javax.microedition.io.SSLProtocolPermission(“ssl://*”)	SSL	Low Level Net Access
javax.microedition.io.Connector.comm	javax.microedition.io.CommProtocolPermission(“comm:*”)	comm	Local Connectivity
javax.microedition.io.PushRegistry	javax.microedition.io.PushRegistryPermission(“*”)	All	Application Auto Invocation

Mapping PDA Optional Packages (JSR 75) Permissions to Function Groups

Table 3: Assigning Permissions Specified in the Personal Information Management Package (of the JSR 75 PDA Optional Packages) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
javax.microedition.pim.ContactList.read	<TBD>	Read User Data Access
javax.microedition.pim.ContactList.write	<TBD>	Write User Data Access
javax.microedition.pim.EventList.read	<TBD>	Read User Data Access

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
javax.microedition.pim.EventList.write	<TBD>	Write User Data Access
javax.microedition.pim.ToDoList.read	<TBD>	Read User Data Access
javax.microedition.pim.ToDoList.write	<TBD>	Write User Data Access
javax.microedition.io.Connector.file.read	<TBD>	Read User Data Access
javax.microedition.io.Connector.file.write	<TBD>	Write User Data Access

The implementation MUST ensure that the user is informed of the nature of the user data to which an application has access (for example, events or to-do lists) before allowing the application access to these functions. Whenever a MIDlet adds, deletes, or updates a PIM entry under the Oneshot permission type, the implementation MUST display the PIM entry to the user for acknowledgement.

Mapping Bluetooth API (JSR 82) Permissions to Function Groups

Table 4: Assigning Proposed Permissions Specified in the Bluetooth API (JSR 82) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
javax.microedition.io.Connector.bluetooth.client	<TBD>	Local Connectivity
javax.microedition.io.Connector.obex.client	<TBD>	Local Connectivity
javax.microedition.io.Connector.obex.client.tcp	<TBD>	Net Access
javax.microedition.io.Connector.bluetooth.server	<TBD>	Local Connectivity
javax.microedition.io.Connector.obex.server	<TBD>	Local Connectivity
javax.microedition.io.Connector.obex.server.tcp	<TBD>	Net Access

Mapping Mobile Media API (JSR 135) Permissions to Function Groups

Table 5: Assigning Permissions Specified in the Mobile Media API (JSR 135) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.media.control.RecordControl`	`javax.microedition.media.PlayerPermission("*", "record")`	Multimedia Recording
`javax.microedition.media.control.VideoControl.getSnapshot`	`javax.microedition.media.PlayerPermission("*", "snapshot")`	Multimedia Recording
`javax.microedition.io.Connector.rtsp`	`javax.microedition.io.RTSPProtocolPermission("rtsp://*")`	Net Access

Mapping Security and Trust Services API (JSR 177) Permissions to Function Groups

Table 6: Assigning Permissions Specified in the Security and Trust Services API (JSR 177) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.apdu.sat`	`javax.microedition.apdu.APDUPermission("sat")`	Assigned permission. Access is *Allowed* for manufacturer and operator domains and *Denied* for other domains.
`javax.microedition.apdu.aid`	`javax.microedition.apdu.APDUPermission("aid")`	Smart Card Communication
`javax.microedition.jcrmi`	`javax.microedition.jcrmi.JCRMIPermission(null)`	Smart Card Communication
`javax.microedition.securityservice.CMSMessageSignatureService`	`javax.microedition.securityservice.SecurityServicePermission("authenticateBinary")`	Authentication

Mapping SIP API (JSR 180) Permissions to Function Groups

Table 7: Assigning Permissions Specified in the SIP API (JSR 180) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.io.Connector.sip`	<TBD>	Call Control
`javax.microedition.io.Connector.sips`	<TBD>	Call Control

Mapping Wireless Messaging API (JSR 205) Permissions to Function Groups

Table 8: Assigning Permissions Specified in the Wireless Messaging API (JSR 205) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Protocol	Function group
`javax.wireless.messaging.sms.send`	<TBD>	SMS	Messaging-Send
`javax.wireless.messaging.sms.receive`	<TBD>	SMS	Messaging-Receive
`javax.microedition.io.Connector.sms`	<TBD>	SMS	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.io.Connector.cbs`	<TBD>	CBS	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.io.Connector.mms`	<TBD>	MMS	Assigned permission. Access is *Allowed* for all protection domains.
`javax.wireless.messaging.cbs.receive`	<TBD>	CBS	Restricted Messaging - Receive
`javax.wireless.messaging.mms.send`	<TBD>	MMS	Messaging-Send
`javax.wireless.messaging.mms.receive`	<TBD>	MMS	Messaging-Receive

Interaction Modes

If the interaction mode is Oneshot for the Messaging-Send, Messaging-Receive, Restricted Messaging-Send or Restricted Messaging-Receive function group, the connections are assigned the Blanket mode as shown in TABLE 9.

Table 9: Interaction Modes for Connections.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group Interaction Mode	Permission Interaction Mode
`javax.microedition.io.Connector.sms`	`<TBD>`	Messaging-Send or Messaging-Receive is Oneshot	Blanket
`javax.microedition.io.Connector.cbs`	`<TBD>`	Restricted Messaging-Send or Restricted Messaging-Receive is Oneshot	Blanket
`javax.microedition.io.Connector.mms`	`<TBD>`	Messaging-Send or Messaging-Receive is Oneshot	Blanket

Mapping Content Handler API (JSR 211) Permissions to Function Groups

Table 10: Assigning Permissions Specified in the Content Handler API (JSR 211) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)s	Function Group
`javax.microedition.content.ContentHandler`	`<TBD>`	Application Auto Invocation

Mapping Advanced Multimedia Supplements API (JSR 234) Permissions to Function Groups

Table 11: Assigning Permissions Specified in the Advanced Multimedia Supplements API (JSR 234) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.amms.control.camera.enableShutterFeedback`	`javax.microedition.amms.AmmsPermission(“*”, “cameraControl.enableShutterFeedback”)`	Multimedia Recording
`javax.microedition.amms.control.tuner.setPreset`	`javax.microedition.amms.AmmsPermission(“*”, “tunerControl.setPreset”)`	Write User Data Access

Mapping Mobile Sensor API (JSR 256) Permissions to Function Groups

Table 12: Assigning Permissions Specified in the Mobile Sensor API to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.sensor.PrivateSensor`	`<TBD>`	Private Sensor Access
`javax.microedition.sensor.ProtectedSensor`	`<TBD>`	Assigned permission. Access is *Allowed* for manufacturer and operator domains and *Denied* for other domains
`javax.microedition.io.Connector.sensor`	`<TBD>`	Assigned permission. Access is *Allowed* for all protection domains.

Mapping Contactless Communication API (JSR 257) Permissions to Function Groups

Table 13: Assigning Permissions Specified in the Contactless Communication API to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.contactless.DiscoveryManager`	`<TBD>`	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.contactless.ndef.NDEFTagConnection.write`	`<TBD>`	NFC Write Access
`javax.microedition.io.Connector.ndef`	`<TBD>`	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.io.Connector.rf`	`<TBD>`	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.io.Connector.sc`	`<TBD>`	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.io.Connector.vtag`	`<TBD>`	Assigned permission. Access is *Allowed* for all protection domains.

Mapping Mobile User Interface Customization API (JSR 258) Permissions to Function Groups

Table 14: Assigning Permissions Specified in the Mobile User Interface Customization API to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
`javax.microedition.theme.manage`	`javax.microedition.theme.ManagePermission`	UI Customization
`javax.microedition.theme.modify`	`javax.microedition.theme.ModifyPermission`	UI Customization

Mapping Mobile Broadcast Service API (JSR 272) Permissions to Function Groups

Table 15: Assigning Permissions Specified in the Mobile Broadcast Service API to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)s	Function Group
`javax.microedition.broadcast.recording.RecordingScheduler.add`	`javax.microedition.broadcast.BroadcastPermission(“*”, “recordingScheduler.add”)`	Mobile Broadcast Service
`javax.microedition.broadcast.recording.RecordingScheduler.remove`	`javax.microedition.broadcast.BroadcastPermission(“*”, “recordingScheduler.remove”)`	Mobile Broadcast Service
`javax.microedition.broadcast.recording.RecordingScheduler.access`	`javax.microedition.broadcast.BroadcastPermission(“*”, “recordingScheduler.access”)`	Mobile Broadcast Service
`javax.microedition.broadcast.esg.ServiceGuide.access`	`javax.microedition.broadcast.BroadcastPermission(“*”, “serviceGuide.access”)`	Mobile Broadcast Service
`javax.microedition.broadcast.ServiceContext.create`	`javax.microedition.broadcast.BroadcastPermission(“*”, “serviceContext.create”)`	Mobile Broadcast Service
`javax.microedition.broadcast.ServiceContext.default`	`javax.microedition.broadcast.BroadcastPermission(“*”, “serviceContext.default”)`	Mobile Broadcast Service
`javax.microedition.broadcast.ServiceContext.select`	`javax.microedition.broadcast.BroadcastPermission(“*”, “serviceContext.select”)`	Mobile Broadcast Service
`javax.microedition.broadcast.purchase.PurchaseObject.purchase`	`javax.microedition.broadcast.BroadcastPermission(“*”, “purchaseObject.purchase”)`	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.broadcast.purchase.PurchaseObject.cancel`	`javax.microedition.broadcast.BroadcastPermission(“*”, “purchaseObject.cancel”)`	Assigned permission. Access is *Allowed* for all protection domains.
`javax.microedition.broadcast.purchase.SubscriptionManager.getSubscriptions`	`javax.microedition.broadcast.BroadcastPermission(“*”, “subscriptionManager.getSubscriptions”)`	Mobile Broadcast Service

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)s	Function Group
javax.microedition.broadcast.ServiceContext.broadcastdatagram	<TBD>	Mobile Broadcast Service
javax.microedition.io.Connector.broadcastfile.read	<TBD>	Mobile Broadcast Service

Mapping IMS Services API (JSR 281) Permissions to Function Groups

Table 16: Assigning Permissions Specified in the IMS Services API to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
javax.microedition.io.Connector.imscore	<TBD>	Net Access
javax.microedition.ims.Media.read	<TBD>	Read User Data Access
javax.microedition.ims.Media.write	<TBD>	Write User Data Access

Mapping Location API 2.0 (JSR 293) Permissions to Function Groups

Table 17:Assigning Permissions Specified in the Location API 2.0 (JSR 293) to Function Groups.

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
javax.microedition.location.Location	javax.microedition.location.LocationPermission(“location”, “location”)	Location
javax.microedition.location.Orientation	javax.microedition.location.LocationPermission(“orientation”, “orientation”)	Location
javax.microedition.location.ProximityListener	javax.microedition.location.LocationPermission(“proximity_listener”, “proximity”)	Location
javax.microedition.location.LandmarkStore.read	javax.microedition.location.LocationPermission(“landmark_store”, “read”)	Landmark
javax.microedition.location.LandmarkStore.write	javax.microedition.location.LocationPermission(“landmark_store”, “write”)	Landmark

Permission Name (MIDP 2.1)	Permission Class (MIDP 3)	Function Group
javax.microedition.location.LandmarkStore.category	javax.microedition.location.LocationPermission(“landmark_store”, “category”)	Landmark
javax.microedition.location.LandmarkStore.management	javax.microedition.location.LocationPermission(“landmark_store”, “management”)	Landmark

8. Recommendations and Guidelines (informative)

The following section(s) provide recommendations and guidelines for various users (developers, implementers, etc.) of this specification.

8.1 Guideline for Applications Referring Non-Mandatory APIs

The MSA Specification includes some conditionally mandatory APIs that do not need to be present, provided a defined condition is not met. When an application is targeted to all MSA devices, it needs to be authored carefully when using APIs that might not be present on some devices. Similar application design recommendations apply when using other APIs that are not part of the current MSA Specification, if the intention is to target a range of devices wider than MSA devices.

Because of implementation options allowed by the Java Virtual Machine Specification and the CLDC Specification, no guaranteed way exists to construct MIDlets in a way that they could execute in the absence of some APIs referenced from the MIDlet. However, the following guideline gives a good probability that the MIDlet can be accepted and executed successfully in a majority of devices.

The main concept in the guideline is to isolate the references to APIs that might not always be present from the main body of the application. Figure 1 illustrates an application and it's internal dependencies and dependencies to parts of the Java platform in the device. The isolation is provided using a Java interface defined in the application (interface A in the figure) and a class implementing this interface (class C in the figure). The main body of the application references only the defined interface. The class implementing the interface (class C in the figure) and possible other application classes referenced from it (the module depicted on the right side of the MIDlet JAR box in the figure) contain the references to the API that might not necessarily be present. This class is instantiated in the main body of the application using `Class.forName()` and `Class.newInstance()`, only after the code of the main body tests the system properties to determine if the API used is present in the device. When the MIDlet executes on a device that does not support the Conditionally Mandatory API, the class implementing the interface is not instantiated or loaded into the virtual machine.

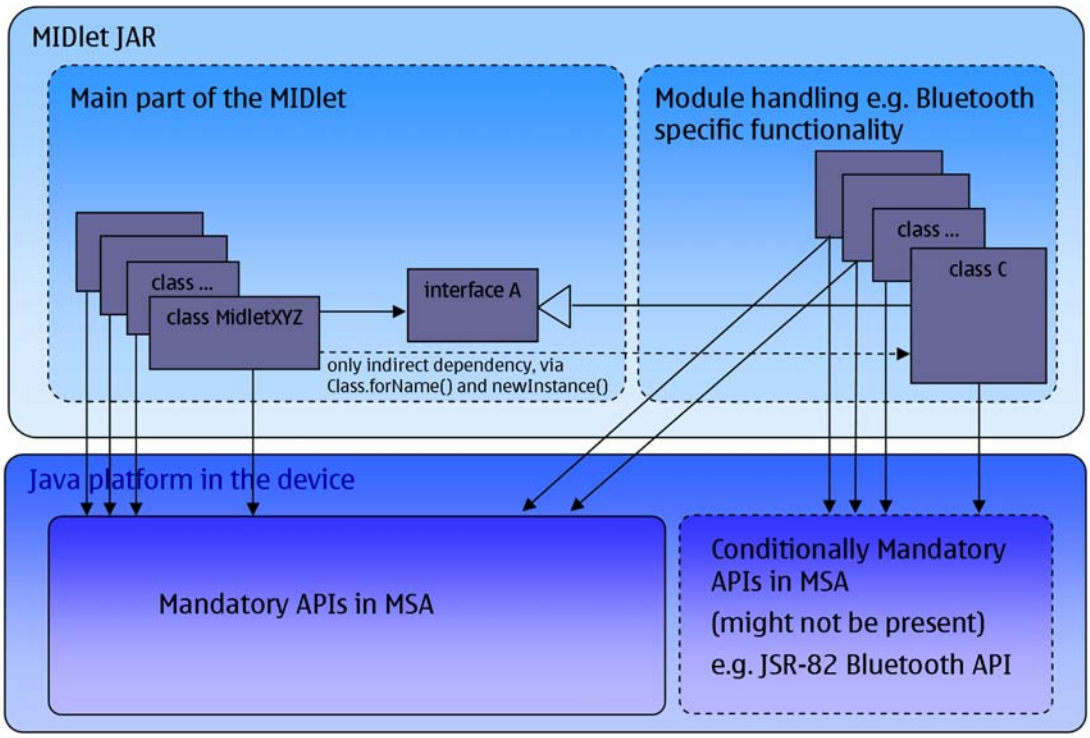


Figure 1. Dependencies between the classes of the MIDlet and the platform

This guidleline is illustrated in the following code example:

```
/* Part of the main body of the application */

public class Foo {

  public void sendViaBluetooth() {

      // Testing if JSR 82 exists on the device

      String btProp =
            java.lang.System.getProperty("bluetooth.api.version");

      if (btProp == null)
         return;

      // JSR 82 exists

      Class c = Class.forName("BluetoothManager");

      BluetoothUsage btu = (BluetoothUsage)c.newInstance();
```

```
      btu.send(".....");

      // ...
  }
}

/* Interface between the main body and the part using JSR 82 */

public interface BluetoothUsage {

    public void send(Object o);

}

/* The class containing the references to JSR 82 */

public class BluetoothManager implements BluetoothUsage {

    public BluetoothManager() {
      return;
    }

    public void send(Object o) {

      // References to JSR-82 only inside this class

      javax.bluetooth.LocalDevice ld =
          javax.bluetooth.LocalDevice.getLocalDevice();

      javax.bluetooth.DiscoveryAgent = ld.getDiscoveryAgent();


      // ...

    }
}
```

9. Roadmap (informative)

The Mobile Service Architecture standardization efforts are intended to be ongoing activities. The goal of MSA is to produce well defined, evolving Java platforms that meet the latest market requirements and the anticipated speed of technology evolution.

The timing of future MSA releases will be determined by the following factors:

- **Time required for mass-market deployment.** The MSA Specification release cycles should be long enough to allow each successive generation of compliant implementations to establish adequate market presence and to ensure the widespread adoption of the specified functionality. However, the release cycles should not be too long so that the latest technologies and features can be introduced in a timely fashion.

- **Availability of relevant Java standards**. The MSA Expert Group itself does not define any new technologies. Rather, the MSA Expert Group relies on the ability of other Java Community Process (JCP) efforts to produce relevant standards that can be adopted by the mobile industry. Therefore, MSA is dependent on the schedules of those JCP efforts that are under development as well as on technology standardization in other relevant standardization organizations.

Any future release of MSA will strive to maintain backward compatibility with existing MSA releases. Each future release will also fulfill the key design goals set for MSA, such as the focus on high-volume devices with limited capabilities, and the requirement to be able to run the specified functionality on the Java ME CDC platform as well.

MSA Specification Leads together with the MSA Expert Group will periodically provide updates on the progress of the MSA specification work and the future plans for MSA.

Appendix A. Summary Tables (informative)

The following sections contain summary information from different parts of this specification to make the specification more readable.

A.1 System Properties

The different JSRs and their clarifications in this specification mention several system properties that can be queried with the method `System.getProperty(String key)`.

A.1.1 API Availability and Version

The following table provides an overview of the properties indicating the presence of a particular API or an Optional Package (and its version):

Property Name	Defined in JSR	Present in		
		MSA 2 EP	MSA 2 SP	MSA 2 AP
`microedition.io.file.FileConnection.version`	75	X	X	X
`microedition.pim.version`	75	X	X	X
`bluetooth.api.version`	82	X	X	X
`obex.api.version`	82	X	X	X
`microedition.profiles`	118, 271 and 139	X	X	X
`microedition.media.version`	135	X	X	X
`microedition.configuration`	139	X	X	X
`wireless.messaging.version`	249	X	X	X
`microedition.msa.version`	249	X	X	X
`xml.jaxp.subset.version`	172 and 249			X
`xml.rpc.subset.version`	172 and 249			X
`microedition.satsa.apdu.version`	177 and 249		X	X
`microedition.satsa.crypto.version`	177 and 249			X
`microedition.satsa.pki.version`	177 and 249			X
`microedition.sip.version`	180			X
`microedition.chapi.version`	211		X	X
`microedition.amms.version`	234		X	X
`microedition.global.version`	238	X	X	X
`microedition.sensor.version`	256	X	X	X
`microedition.contactless.version`	257		X	X
`microedition.theme.version`	258		X	X
`microedition.broadcast.version`	272		X	X

Property Name	Defined in JSR	Present in		
		MSA 2 EP	MSA 2 SP	MSA 2 AP
javax.microedition.xmlapi.version	280		X	X
javax.microedition.xmlapi.events.version	280		X	X
javax.microedition.ims.version	281			X
microedition.m2g.version	287		X	X
microedition.m2g.svg.baseProfile	287		X	X
microedition.m2g.svg.version	287		X	X
microedition.m2g.vectorgraphics.version	287		X	X
fluid.version	290			X
microedition.location.version	293		X	X
microedition.m3g.version	297		X	X

A.1.2 Other System Properties

The following table summarizes all other properties specified in the MSA 2 component JSRs or in the additional clarifications:

Property Name	Defined in JSR	Present in		
		MSA 2 EP	MSA 2 SP	MSA 2 AP
file.separator	75	X	X	X
fileconn.dir.photos	249	X	X	X
fileconn.dir.videos	249	X	X	X
fileconn.dir.graphics	249	X	X	X
fileconn.dir.tones	249	X	X	X
fileconn.dir.music	249	X	X	X
fileconn.dir.recordings	249	X	X	X
fileconn.dir.private	249	X	X	X
fileconn.dir.roots.external	249	X	X	X
fileconn.dir.photos.name	249	X	X	X
fileconn.dir.videos.name	249	X	X	X
fileconn.dir.graphics.name	249	X	X	X
fileconn.dir.tones.name	249	X	X	X
fileconn.dir.music.name	249	X	X	X
fileconn.dir.recordings.name	249	X	X	X
fileconn.dir.roots.names	249	X	X	X
fileconn.dir.private.name	249	X	X	X
bluetooth.l2cap.receiveMTU.max	82	X	X	X

Property Name	Defined in JSR	Present in		
		MSA 2 EP	MSA 2 SP	MSA 2 AP
bluetooth.connected.devices.max	82	X	X	X
bluetooth.connected.inquiry	82	X	X	X
bluetooth.connected.page	82	X	X	X
bluetooth.connected.inquiry.scan	82	X	X	X
bluetooth.connected.page.scan	82	X	X	X
bluetooth.master.switch	82	X	X	X
bluetooth.sd.trans.max	82	X	X	X
bluetooth.sd.attr.retrievable.max	82	X	X	X
microedition.deviceid	271		X	X
microedition.subscriberid	271		X	X
microedition.locale	118 and 271	X	X	X
microedition.commports	118, 271 and 249	X	X	X
microedition.hostname	118 and 271	X	X	X
supports.mixing	135	X	X	X
supports.audio.capture	135	X	X	X
supports.video.capture	135	X	X	X
supports.recording	135	X	X	X
audio.encodings	135	X	X	X
video.encodings	135	X	X	X
video.snapshot.encodings	135	X	X	X
streamable.contents	135	X	X	X
microedition.platform	139 and 249	X	X	X
microedition.encoding	139 and 271	X	X	X
wireless.messaging.sms.smsc	205	X	X	X
wireless.messaging.mms.mmsc	205	X	X	X
microedition.smartcardslots	177		X	X
supports.mediacapabilities	234		X	X
tuner.modulations	234		X	X
audio.samplerates	234		X	X
audio3d.simultaneouslocations	234		X	X
camera.orientations	234		X	X
camera.resolutions	234		X	X
microedition.theme.modifysupported	258		X	X
microedition.theme.platform	258		X	X
microedition.theme.applicationspecific	258		X	X

Property Name	Defined in JSR	Present in		
		MSA 2 EP	MSA 2 SP	MSA 2 AP
microedition.broadcast.supports.overlay	272		X	X
microedition.broadcast.supports.timedrecording	272		X	X
microedition.broadcast.supports.filecache	272		X	X
microedition.broadcast.supports.purchasing	272		X	X
microedition.location.areamonitoring	293		X	X
microedition.location.orientation	293		X	X

A.2 Network Protocols and Content Formats

This section contains summary tables of network protocols and content formats. The tables indicate whether the network protocol or content format is mandatory or optional for MSA 2 compliant implementations. The tables distinguish between items that are mandatory for MSA 2 Advanced Platform, MSA 2 Standard Platform and MSA 2 Entry Platform.

Note: These tables do not list all network protocols and content formats that might be supported by implementations, but only those optional network protocols and formats that are explicitly mentioned in this specification.

A.2.1 Network Protocols

The following table lists network protocols that can be used for communication in MSA 2 compliant implementations, and whether the protocol is mandatory or optional. Refer to the legend below the table.

Protocol	Java ME API	JSR	MSA 2 EP	MSA 2 SP	MSA 2 AP
HTTP client	`javax.microedition.io.HttpConnection`	118/271	M	M	M
HTTP over SSLv3.0 (HTTPS) client	`javax.microedition.io.HttpsConnection`	118/271	M*	M	M
HTTP over TLS v1.0 (HTTPS) client	`javax.microedition.io.HttpsConnection`	118/271	O	O	O
TCP socket client	`javax.microedition.io.SocketConnection`	118/271	M	M	M
TCP socket server	`javax.microedition.io.ServerSocketConnection`	118/271	O	O	O
TLS v1.0 client	`javax.microedition.io.SecureConnection`	118/271	O	O	O
SSL v3.0 client	`javax.microedition.io.SecureConnection`	118/271	M*	M*	M
UDP	`javax.microedition.io.UDPDatagramConnection`	118/271	O	O	O
SIP	`javax.microedition.sip.SIPConnection`	180	-	-	M

* An implementation that supports TLS v1.0 and supports section E of RFC 2246 (to provide backward compatibility with SSL v3.0 servers) is compliant with this requirement.

Protocol	Java ME API	JSR	MSA 2 EP	MSA 2 SP	MSA 2 AP
SMS	`javax.wireless.messaging.MessageConnection`	120/205	M	M	M
MMS	`javax.wireless.messaging.MessageConnection`	205	M	M	M
CBS	`javax.wireless.messaging.MessageConnection`	120/205	O	O	O
Bluetooth L2CAP client	`javax.bluetooth.L2CAPConnection`	82	C	C	C
Bluetooth L2CAP server	`javax.bluetooth.L2CAPConnectionNotifier`	82	C	C	C
Bluetooth RFCOMM client	`javax.microedition.io.StreamConnection`	82	C	C	C
Bluetooth RFCOMM server	`javax.microedition.io.StreamConnectionNotifier`	82	C	C	C
Bluetooth OBEX client	`javax.obex.ClientSession`	82	C	C	C
Bluetooth OBEX server	`javax.obex.SessionNotifier`	82	C	C	C
IrDA OBEX client	`javax.obex.ClientSession`	82	C	C	C
IrDA OBEX server	`javax.obex.SessionNotifier`	82	C	C	C
APDU	`javax.microedition.apdu.APDUConnection`	177	O	C	C
Java Card RMI	`javax.microedition.jcrmi.JavaCardRMIConnection`	177	O	O	O
SIP client	`javax.microedition.sip.SipClientConnection`	180			M
SIP cerver	`javax.microedition.sip.SipServerConnection`	180			M
ISO 14443	`javax.microedition.contactless.sc.ISO14443Connection`	257		C	C
NDEF tag	`javax.microedition.contactless.ndef.NDEFTagConnection`	257		C	C
Plain tag	`javax.microedition.contactless.rf.PlainTagConnection`	257		C	C
Visual tag	`javax.microedition.contactless.visual.VisualTagConnection`	257		C	C
Broadcast datagram	`javax.microedition.broadcast.connection.BroadcastDatagramConnection`	272		C	C
Broadcast file	`javax.microedition.broadcast.connection.BroadcastFileConnection`	272		C	C
IMS core service	`javax.microedition.ims.core.CoreService`	281			C

Notation: **M** – Mandatory **O** – Optional **C** – Conditionally Mandatory

A.2.2 Content Formats

The following table lists content formats that are used for presentation in MSA 2 compliant implementations and whether the format is mandatory or optional. Refer to the legend below the table.

Content format	Java API	JSR	MSA 2 EP	MSA 2 SP	MSA 2 AP
PNG	`javax.microedition.lcdui.Image`	118/ 271	M	M	M
PNG	`javax.microedition.m3g.Image2D`	297		M	M
JPEG (JFIF)	`javax.microedition.lcdui.Image`	118/ 271	M	M	M
JPEG (JFIF)	`javax.microedition.media.Player` (video snapshot encoding)	135, 249	C	C	C
JPEG (JFIF)	`javax.microedition.m3g.Image2D`	297		M	M
M3G	`javax.microedition.m3g.Object3D`	297		M	M
SVG Tiny 1.1	`javax.microedition.m2g.SVGImage`	287		M	M
8 kHz, 8-bit linear PCM audio in WAV	`javax.microedition.media.Player`	135	C	M	M
AMR-NB*	`javax.microedition.media.Player`	135	C	M	M
AMR-WB	`javax.microedition.media.Player`	135		R	R
MP3	`javax.microedition.media.Player`	135	R	M	M
MIDI	`javax.microedition.media.Player`	135	M	M	M
SP-MIDI	`javax.microedition.media.Player`	135	M	M	M
Tone sequence	`javax.microedition.media.Player`	135, 249	M	M	M
3GP	`javax.microedition.media.Player`	135	C	M	M
MP4	`javax.microedition.media.Player`	135		M	M

Notation: **M** – Mandatory
C – Conditionally Mandatory
R – Recommended

* AMR-NB support is NOT REQUIRED for compliant *development tools*, *device emulators*, or a *reference implementation* running on a device emulator.

A.3 Hardware Requirements and Recommendations

The following table summarizes the hardware-related requirements and recommendations defined in this specification:

Feature	EP		SP		AP	
	Req.	Rec.	Req.	Rec.	Req.	Rec.
Minimum supported screen size	None	128x128	None	176x220, 220x176	None	240x320, 320x240
Minimum supported number of colors	None	4096	None	65536	None	262144
Pixel aspect ratio	None	1:1	None	1:1	None	1:1
Minimum heap size available to MIDlet	512 kB	600 kB	1024 kB	2048 kB	2048 kB	4096 kB
Minimum supported clock resolution	None	40 ms	None	40 ms	None	40 ms
Minimum supported number of application-created threads per MIDlet	10	None	10	None	10	None
Minimum supported MIDlet Suite/LIBlet JAR file download size	150 kB	300 kB	512 kB	1024 kB	1024 kB	2048 kB
Minimum supported MIDlet Suite/LIBlet JAR file install size	150 kB	300 kB	512 kB	1024 kB	1024 kB	2048 kB
Minimum supported MIDlet suite/LIBlet JAD size	10 kB	None	20 kB	None	20 kB	None
Minimum supported size of an individual attribute value in JAD/Manifest	2048 B	None	2048 B	None	2048 B	None
Minimum supported size of an individual attribute name in JAD/Manifest	70 B	None	70 B	None	70 B	None
Minimum supported number of individual attributes in JAD/Manifest	100	None	512	None	512	None
Minimum supported number of MIDlets in a MIDlet suite	5	None	5	None	5	None

Feature	EP		SP		AP	
	Req.	Rec.	Req.	Rec.	Req.	Rec.
Minimum supported RMS data size per MIDlet suite	65536 (64 kB)	None	262144 (256 kB)	None	524288 (512 kB)	None
Minimum supported number of record stores owned by a MIDlet suite	10	None	10	None	10	None
Minimum supported image object size	Size of full screen full depth image but at least 32768 B	None	Size of full screen full depth image but at least 77440 B	None	Size of full screen full depth image but at least 172800 B	None
Minimum supported PNG color depth	32 bit per pixel	None	32 bit per pixel	None	32 bit per pixel	None
Minimum supported timer resolution	None	40 ms	None	40 ms	None	40 ms
Minimum supported number of timers per MIDlet	None	5	None	5	None	5

These requirements and recommendations apply to all MSA 2 compliant implementations.

A.4 Security Permissions and Function Groups

The following table summarizes the Function Groups and permissions belonging to each Function Group.

Function Groups with associated Permissions	API	MSA 2 EP	MSA 2 SP	MSA 2 AP
Net Access				
javax.microedition.io.Connector.http	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.https	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.obex.client.tcp	Bluetooth API	X	X	X
javax.microedition.io.Connector.obex.server.tcp	Bluetooth API	X	X	X
javax.microedition.io.Connector.imscore	IMS API			X
javax.microedition.io.Connector.rtsp	Mobile Media API		X	X
Low Level Net Access				
javax.microedition.io.Connector.datagram	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.datagramreceiver	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.socket	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.serversocket	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.ssl	MIDP 2.1/3.0	X	X	X
Call Control				
javax.microedition.io.Connector.sip	SIP API			X
javax.microedition.io.Connector.sips	SIP API			X
Local Connectivity				
javax.microedition.io.Connector.comm	MIDP 2.1/3.0	X	X	X
javax.microedition.io.Connector.bluetooth.client	Bluetooth API	X	X	X
javax.microedition.io.Connector.obex.client	Bluetooth API	X	X	X
javax.microedition.io.Connector.bluetooth.server	Bluetooth API	X	X	X

Function Groups with associated Permissions	API	MSA 2 EP	MSA 2 SP	MSA 2 AP
javax.microedition.io.Connector.obex.server	Bluetooth API	X	X	X
Application Auto Invocation				
javax.microedition.io.PushRegistry	MIDP 2.1/3.0	X	X	X
javax.microedition.content.ContentHandler	Content Handler API		X	X
Read User Data Access				
javax.microedition.pim.ContactList.read	PDA Optional Packages	X	X	X
javax.microedition.pim.EventList.read	PDA Optional Packages	X	X	X
javax.microedition.pim.ToDoList.read	PDA Optional Packages	X	X	X
javax.microedition.io.Connector.file.read	PDA Optional Packages	X	X	X
javax.microedition.ims.Media.read	IMS API			X
Write User Data Access				
javax.microedition.pim.ContactList.write	PDA Optional Packages	X	X	X
javax.microedition.pim.EventList.write	PDA Optional Packages	X	X	X
javax.microedition.pim.ToDoList.write	PDA Optional Packages	X	X	X
javax.microedition.io.Connector.file.write	PDA Optional Packages	X	X	X
javax.microedition.amms.control.tuner.setPreset	AMMS API		X	X
javax.microedition.ims.Media.write	IMS API			X
Multimedia Recording				
javax.microedition.media.control.RecordControl	Mobile Media API	X	X	X
javax.microedition.media.control.VideoControl.getSnapshot	Mobile Media API	X	X	X
javax.microedition.amms.control.camera.enableShutterFeedback	AMMS API		X	X

Function Groups with associated Permissions	API	MSA 2 EP	MSA 2 SP	MSA 2 AP
Smart Card Communication				
javax.microedition.apdu.aid	SATSA API		X	X
javax.microedition.jcrmi	SATSA API			X
Authentication				
javax.microedition.securityservice.CMSMessageSignatureService	SATSA API			X
Location				
javax.microedition.location.Location	Location API		X	X
javax.microedition.location.Orientation	Location API		X	X
javax.microedition.location.ProximityListener	Location API		X	X
Landmark				
javax.microedition.location.LandmarkStore.read	Location API		X	X
javax.microedition.location.LandmarkStore.write	Location API		X	X
javax.microedition.location.LandmarkStore.category	Location API		X	X
javax.microedition.location.LandmarkStore.management	Location API		X	X
Messaging-Send				
javax.wireless.messaging.sms.send	WMA API	X	X	X
javax.wireless.messaging.mms.send	WMA API	X	X	X
Messaging-Receive				
javax.wireless.messaging.sms.receive	WMA API	X	X	X
javax.wireless.messaging.mms.receive	WMA API	X	X	X
Restricted Messaging-Receive				
javax.wireless.messaging.cbs.receive	WMA API	X	X	X
Private Sensor Access				

Function Groups with associated Permissions	API	MSA 2 EP	MSA 2 SP	MSA 2 AP
javax.microedition.sensor.PrivateSensor	Sensor API	X	X	X
NFC Write Access				
javax.microedition.contactless.ndef.NDEFTagConnection.write	Contactless Communication API		X	X
UI Customization				
javax.microedition.theme.manage	UI Customization API		X	X
javax.microedition.theme.modify	UI Customization API		X	X
Mobile Broadcast Service				
javax.microedition.broadcast.recording.RecordingScheduler.add	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.recording.RecordingScheduler.remove	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.recording.RecordingScheduler.access	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.esg.ServiceGuide.access	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.ServiceContext.create	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.ServiceContext.default	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.ServiceContext.select	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.purchase.SubscriptionManager.getSubscriptions	Mobile Broadcast Service API		X	X
javax.microedition.broadcast.ServiceContext.broadcastdatagram	Mobile Broadcast Service API		X	X

Function Groups with associated Permissions	API	MSA 2 EP	MSA 2 SP	MSA 2 AP
`javax.microedition.io.Connector.broadcastfile.read`	Mobile Broadcast Service API		X	X

Appendix B. List of Java ME JSRs (informative)

The following table lists all known Java ME JSRs with a brief comment on their status in the MSA 2 specification.

JSR#	JSR Name	JSR 249 component	Comment
1	Real-time Specification for Java		Not in MSA scope
30	J2ME™ Connected, Limited Device Configuration		Replaced by JSR 139
36	Connected Device Configuration		Replaced by JSR 218
37	Mobile Information Device Profile for the J2ME™ Platform		Replaced by JSR 118
46	Foundation Profile		Replaced by JSR 219
50	Distributed Real-Time Specification		Not in MSA scope
62	Personal Profile Specification		Replaced by JSR 216
66	RMI Optional Package Specification Version 1.0		Requires CDC that is not required in MSA 2
68	J2ME™ Platform Specification		N/A
75	PDA Optional Packages for the J2ME™ Platform	yes	Since MSA 1.0
82	Java™ APIs for Bluetooth	yes	Since MSA 1.0
99	Java Specification Participation Agreement		N/A
113	Java™ Speech API 2.0		Not included in MSA 2.0
118	Mobile Information Device Profile 2.1	yes	Basis for MSA 2 EP
120	Wireless Messaging API		Replaced by JSR 205
129	Personal Basis Profile Specification		Replaced by JSR 217
135	Mobile Media API	yes	Since MSA 1.0
139	Connected Limited Device Configuration 1.1	yes	Since MSA 1.0
169	JDBC Optional Package for CDC/Foundation Profile		Requires CDC that is not required in MSA 2
171	Java Community Process℠ (JCP) Program, version 2.5		N/A
172	J2ME™ Web Services Specification	yes	JAX RPC optional package only. Other packages not included.
177	Security and Trust Services API for J2ME	yes	Since MSA 1.0
179	Location API for J2ME™		Replaced by JSR 293
180	SIP API for J2ME™	yes	Since MSA 1.0
184	Mobile 3D Graphics API for J2ME™		Replaced by JSR 297
185	Java™ Technology for the Wireless Industry		Most JTWI requirements included in the MSA spec
186	Presence		Not included in MSA 2.0
187	Instant Messaging		Not included in MSA 2.0
190	Event Tracking API for J2ME		Not included in MSA 2.0
195	Information Module Profile		N/A
205	Wireless Messaging API 2.0	yes	Since MSA 1.0
209	Advanced Graphics and User Interface Optional Package for the J2ME Platform		Requires CDC that is not required in MSA 2

211	Content Handler API	yes	Since MSA 1.0
215	Java Community Process℠ version 2.6		N/A
216	Personal Profile 1.1		Requires CDC that is not required in MSA 2
217	Personal Basis Profile 1.1		Requires CDC that is not required in MSA 2
218	Connected Device Configuration (CDC) 1.1		CDC is not required in MSA 2, but it can be used in an implementation underneath MIDP as specified in MIDP specifications
219	Foundation Profile 1.1		Requires CDC that is not required in MSA 2
226	Scalable 2D Vector Graphics API for J2ME		Replaced by JSR 287
228	Information Module Profile - Next Generation (IMP-NG)		N/A
229	Payment API		Issues with TCK/RI support and licensing
230	Data Sync API		Issues with TCK/RI support and licensing
232	Mobile Operational Management		Requires CDC that is not required in MSA 2
234	Advanced Multimedia Supplements	yes	Since MSA 1.0
238	Mobile Internationalization API	yes	Since MSA 1.0
239	Java™ Binding for the OpenGL® ES API	yes	Included in MSA 2.0
242	Digital Set Top Box Profile - "On Ramp to OCAP"		Not in MSA scope
246	Device Management API		Issues with TCK/RI support and licensing
248	Mobile Service Architecture		JSR 249 is backward compatible with JSR 248
249	Mobile Service Architecture 2		N/A
253	Mobile Telephony API (MTA)		Issues with TCK/RI support and licensing
256	Mobile Sensor API	yes	Included in MSA 2.0
257	Contactless Communication API	yes	Included in MSA 2.0
258	Mobile User Interface Customization API	yes	Included in MSA 2.0
259	Ad Hoc Networking API		Issues with TCK/RI support and licensing
266	Unified Message Box Access API (UMBA-API)		Issues with TCK/RI support and licensing
271	Mobile Information Device Profile 3	yes	Basis for MSA 2 SP and AP
272	Mobile Broadcast Service API for Handheld Terminals	yes	Included in MSA 2.0
278	Resource Management API for Java™ ME		Not in scope. The API is for system programmers, not application developers.
279	Service Connection API for Java™ ME		Not included in MSA 2.0
280	XML API for Java™ ME	yes	Included in MSA 2.0
281	IMS Services API	yes	Included in MSA 2.0
282	RTSJ version 1.1		Not in MSA scope
287	Scalable 2D Vector Graphics API 2.0 for	yes	Included in MSA 2.0

	Java ME		
290	Java™ Language & XML User Interface Markup Integration	yes	Included in MSA 2.0
293	Location API 2.0	yes	Replaces JSR 179 from MSA 1.0
297	Mobile 3D Graphics API 2.0	yes	Replaces JSR 184 from MSA 1.0
298	Telematics API for Java™ ME		Not in MSA scope
300	DRM API for Java™ ME		Not included in MSA 2.0
302	Safety Critical Java™ Technology		Not in MSA scope
304	Mobile Telephony API version 2		Issues with TCK/RI support and licensing
306	Towards a new version of the JCP		N/A
307	Network Mobility and Mobile Data API		Not included in MSA 2.0
320	Services Framework		Not included in MSA 2.0
909	Java™ Specification Participation Agreement		N/A
913	JCP 2.0		Not in MSA scope
927	Java TV™ API 1.1		Not in MSA scope