Mobile Service Architecture Specification

Version 1.00 – 27-September-2006

Java Community ProcessSM
Java™ Platform, Micro Edition

NOKIA

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

vodafone

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

JSR 248 Expert Group
jsr-248-comments@jcp.org

RESEARCH AND EVALUATION LICENSE

The JSR-248 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 248 - Mobile Service Architecture ("Specification")

Version:	1.00
Status:	Final Specification
Specification Lead:	Nokia Corporation, Vodafone Group Services Limited
Release:	September 27th, 2006

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.

Contents:

1. SCOPE 7
2. REFERENCES 8
2.1 NORMATIVE REFERENCES 8
3. TERMINOLOGY AND CONVENTIONS 11
3.1 CONVENTIONS 11
3.2 DEFINITIONS 11
3.3 ABBREVIATIONS 11
3.4 TYPOGRAPHIC CONVENTIONS 12
3.5 APPROVED VERSION HISTORY 12
3.6 CURRENT VERSION HISTORY 12
3.7 FEEDBACK 13
4. INTRODUCTION (INFORMATIVE) 14
4.1 DESIGN GOALS 14
4.2 SPECIFICATION STRUCTURE 15
4.3 EXPERT GROUP 15
5. MSA COMPONENT JSRS (NORMATIVE) 17
6. ADDITIONAL CLARIFICATIONS (NORMATIVE) 21
6.1 J2ME CONNECTED LIMITED DEVICE CONFIGURATION (JSR 139) 21
6.2 MOBILE INFORMATION DEVICE PROFILE (JSR 118) 25
6.3 PDA OPTIONAL PACKAGES FOR THE J2ME PLATFORM (JSR 75) 32
6.4 JAVA APIS FOR BLUETOOTH (JSR 82) 45
6.5 MOBILE MEDIA API (JSR 135) 47
6.6 J2ME WEB SERVICES (JSR 172) 52
6.7 SECURITY AND TRUST SERVICES API (JSR 177) 54
6.8 LOCATION API FOR J2ME (JSR 179) 58
6.9 SIP API FOR J2ME (JSR 180) 59
6.10 MOBILE 3D GRAPHICS API (JSR 184) 63
6.11 WIRELESS MESSAGING API (JSR 205) 66
6.12 CONTENT HANDLER API (JSR 211) 71
6.13 SCALABLE 2D VECTOR GRAPHICS API FOR J2ME (JSR 226) 72
6.14 PAYMENT API (JSR 229) 74
6.15 ADVANCED MULTIMEDIA SUPPLEMENTS (JSR 234) 75
6.16 MOBILE INTERNATIONALIZATION API (JSR 238) 79
7. ADDITIONAL REQUIREMENTS (NORMATIVE) 80
7.1 REQUIREMENTS INHERITED FROM JTWI 1.0 (JSR 185) 80
7.2 HARDWARE REQUIREMENTS 92
7.3 SECURITY REQUIREMENTS 94
8. RECOMMENDATIONS AND GUIDELINES (INFORMATIVE) 107
8.1 GUIDELINE FOR APPLICATIONS REFERRING NON-MANDATORY APIS 107
9. ROADMAP (INFORMATIVE) 110
APPENDIX A. SUMMARY TABLES (INFORMATIVE) 111
A.1 SYSTEM PROPERTIES 111
A.2 NETWORK PROTOCOLS AND CONTENT FORMATS 114

A.3 HARDWARE REQUIREMENTS AND RECOMMENDATIONS 116

1. Scope

This Mobile Service Architecture (MSA) Specification (JSR 248) defines the next step in the Java platform evolution for mobile handsets, based on the Connected Limited Device Configuration (CLDC) of the Java™ Platform, Micro Edition (Java ME). Additionally, the MSA Specification defines an MSA Subset for devices with limited resources. MSA Subset is a pure subset of MSA and, as such, MSA is fully backwards compatible with MSA Subset. Both MSA and MSA Subset can also be implemented using Connected Device Configuration (CDC) as long as the implementation complies with this specification.

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

2. References

2.1 Normative References

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

[IRDA] Infrared Data Organization, http://www.irda.org/, http://irda.affiniscape.com/displaycommon.cfm?an=1&subarticlenbr=7

[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, 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 1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=135

[JSR139] “Connected Limited Device Configuration”, Version 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, 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 1.0.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=180

[JSR184] “Mobile 3D Graphics API for J2ME”, Version 1.1, Java Community Process, http://www.jcp.org/en/jsr/detail?id=184

[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

[JSR226] “Scalable 2D Vector Graphics API for J2ME”, Version 1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=226

[JSR229] "Payment API", Version 1.1.0, Java Community Process, http://www.jcp.org/en/jsr/detail?id=229

[JSR234] "Advanced Multimedia Supplements", Version 1.0, 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

[PKCS] PKCS#15 v.1.1, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-15/

[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

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

[RFC2234] "Augmented BNF for Syntax Specifications: ABNF", D. Crocker, Ed., P. Overell, November 1997, URL:http://www.ietf.org/rfc/rfc2234.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

[RFC2716] "PPP EAP TLS Authentication Protocol", B. Aboba, D. Simon, October 1999, URL:http://www.ietf.org/rfc/rfc2716.txt

[RFC2976] "The SIP INFO Method ", S. Donovan, October 2000, http://www.ietf.org/rfc/rfc2976.txt

[RFC3261] "SIP: Session Initiation Protocol ", J. Rosenberg, H. Schulzrinne, G. Gamarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, June 2002, http://www.ietf.org/rfc/rfc3261.txt

[RFC3262] "Reliability of Provisional Responses in the Session Initiation Protocol (SIP)", J. Rosenberg, H. Schulzrinne, June 2002, http://www.ietf.org/rfc/rfc3262.txt

[RFC3265] "Session Initiation Protocol (SIP)-Specific Event Notification", A. B. Roach, June 2002, URL:http://www.ietf.org/rfc/rfc3265.txt

[RFC3311] "The Session Initiation Protocol (SIP) UPDATE Method", J. Rosenberg, September 2002, URL:http://www.ietf.org/rfc/rfc3311.txt

[RFC3428] "Session Initiation Protocol (SIP) Extension for Instant Messaging ", B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, D. Gurle, December 2002, URL:http://www.ietf.org/rfc/rfc3428.txt

[RFC3515] "The Session Initiation Protocol (SIP) Refer Method", R. Sparks, April 2003, URL:http://www.ietf.org/rfc/rfc3515.txt

[RFC3903] "Session Initiation Protocol (SIP) Extension for Event State Publication", A. Niemi, October 2004, URL:http://www.ietf.org/rfc/rfc3903.txt

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

[SCPROV] WAP Smart Card Provisioning (SCPROV), WAP-186-ProvSC-20010710-

a, 2001, http://www.wapforum.org/what/technical.htm

[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

[SVG] Scalable Vector Graphics (SVG) 1.1 Specification, W3C, 2003, http://www.w3.org/TR/2003/REC-SVG11-20030114/

[SVG Mobile] Mobile SVG Profiles: SVG Tiny and SVG Basic, W3C, 2003, http://www.w3.org/TR/2003/REC-SVGMobile-20030114/

[WIM] WAP Wireless Identity Module Specification (WIM), WAP-260-WIM-20010712-a, 2001, http://www.wapforum.org/what/technical.htm

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) referred to in the MSA Specification.
Conditionally mandatory	MUST be implemented according to the specification if the given condition is fulfilled.
Mandatory	MUST be implemented according to the specification.
MSA compliant implementation	Implementation that conforms to the requirements of this specification.
MSA compliant device	A device that has an implementation that conforms to the requirements of this specification.
MSA 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 TCK delivery package.

3.3 Abbreviations

CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
J2ME	Java 2 Platform, Micro Edition
Java ME	Java Platform, Micro Edition
JCP	Java Community Process

JSR	Java Specification Request
MIDP	Mobile Information Device Profile
MSA	Mobile Service Architecture
OMA	Open Mobile Alliance
RI	Reference Implementation
TCK	Technology Compatibility Kit
UI	User Interface

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
n/a	n/a	No prior version

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.

3.7 Feedback

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

jsr-248-comments@jcp.org

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

MSA accepts *comments* and *clarification proposals* (Contributions) through two main channels: 1) directly from the MSA Expert Group members, and 2) from the MSA 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 Specification section that it addresses, and other relevant information. These Open Issues are then discussed by the MSA Expert Group. As a result of the discussions, possible new clarifications might be generated and added to the MSA Specification.

MSA 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 clarification or clarifications in the component JSR specification itself. If this happens within a schedule that is reasonable from the viewpoint of the MSA release schedule, MSA references the Maintenance Release directly, and does not include the clarification or clarifications in the MSA 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 Specification release then references the Maintenance Release of the component JSR, and removes the clarification from the MSA Specification and removes the corresponding test or tests from the MSA TCK.

Important: All contributions sent to the MSA Expert Group (comments, clarification proposals, and so on) are, by default, also contributions to the respective component JSR or JSRs. The MSA 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 Java Technology for the Wireless Industry (JTWI) initiative. Mobile Service Architecture Initiative continues the work by introducing two new JSRs to cover the whole range of platform technologies needed by the mobile industry today.

To address the market needs for high-volume devices that can support the Java platform, JTWI (JSR 185) focused on mobile devices with limited resources and capabilities. MSA Specification (JSR 248) addresses an even broader set of devices with more enhanced and diverse capabilities but continues its focus on high-volume mobile devices. This JSR broadens the architecture defined by JTWI 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 old JTWI environment and with the future MSA Advanced environment defined by the Mobile Service Architecture Advanced activity (JSR 249).

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

4.1 Design Goals

The primary design goal of the MSA 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 contains two sets of mandatory component JSRs, additional clarifications, additional requirements and recommendations for implementers. The specification guides platform implementers in their efforts to maximize the interoperability of their implementation with other MSA compliant implementations and application developers in their efforts to make sure that their applications will work in wide variety of MSA compliant devices.

The second design goal for MSA is to enable its use in a wide variety of different markets and customer segments. This is achieved by introducing two platform definitions (MSA and MSA Subset), support for the possibility to use also CDC as the underlying configuration, and well-defined conditionality for features that may not be available on all MSA compliant devices.

The third design goal for MSA is to ensure highest level of consistency in the definition of both MSA and the upcoming MSA Advanced environment. This goal was also reflected in the selection of the common Expert Group (EG) for both JSR 248 and JSR 249. The MSA Expert Group has visibility to both specifications and is able to guide the work accordingly.

Special attention was paid to the following two objectives when selecting the mandatory component JSRs:

- Build a feature-rich platform that reflects the market needs from 2006 onwards
- Choose only component JSRs that are in final status by May 2006.

4.2 Specification Structure

The MSA Specification consists of the following main logical elements:

- **Mandatory and Conditionally Mandatory Component JSRs.** The MSA Specification describes the essential client components of an end-to-end wireless environment. It defines two sets of component JSRs: *MSA* and *MSA Subset*. MSA Subset is a proper subset of MSA component JSRs, providing forward compatibility for applications written for MSA Subset.
- **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 JTWI, security, supported content formats, and so on. Additional requirements provide implementers with more requirements and consequently improve backwards compatibility, interoperability, and predictability of MSA compliant implementations.
- **Recommendations and Guidelines.** These are suggestions for developers and implementers of this specification on how to write applications for MSA environment and create an optimal MSA compliant implementation.
- **Roadmap.** The MSA roadmap aims to describe the future view of the mobile Java platform. The separately released roadmap document contains tentative proposals for the content and timing of future versions of MSA specifications.

4.3 Expert Group

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

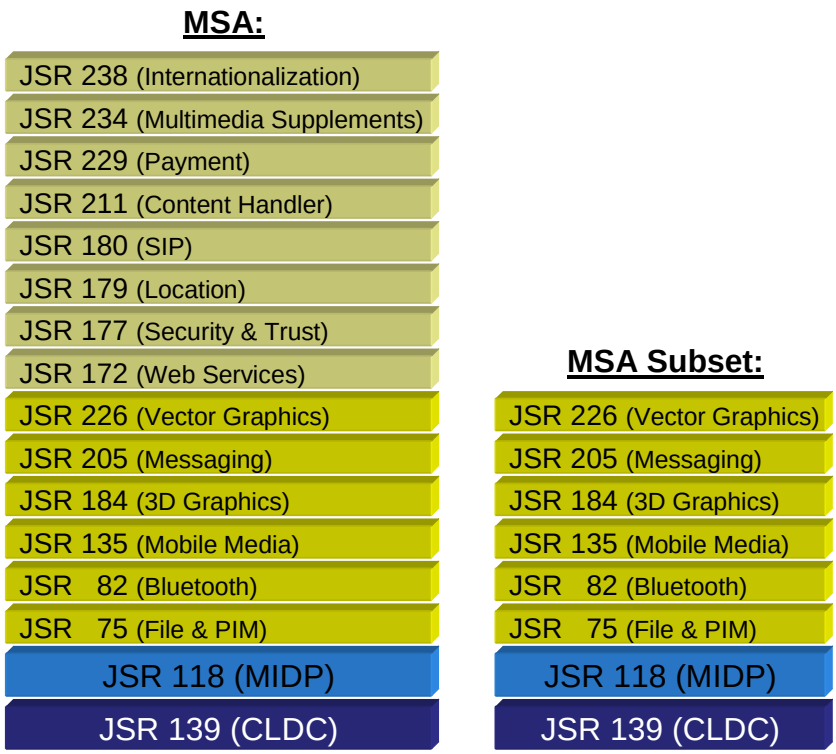
Following are the members of the MSA Expert Group:

- BEA Systems
- BenQ
- Cingular Wireless
- Motorola
- Nokia Corporation

- NTT DoCoMo
- Orange France SA
- Research In Motion
- Samsung
- Siemens
- Sony Ericsson Mobile Communications AB
- Sprint
- Sun Microsystems, Inc.
- T-Mobile International AG & Co. KG
- Vodafone Group Services Limited

5. MSA Component JSRs (normative)

This specification defines two platforms: *MSA* and *MSA Subset*. The following picture shows the content of these two platforms:



A compliant implementation MUST conform to the requirements for either MSA or MSA Subset:

- **MSA**:
    1. Implement every component JSR as specified in the following table.
    2. Comply with all the additional requirements per component JSR as specified in the relevant Additional Clarifications sections.
    3. Comply with all the additional requirements as specified in the Additional Requirements chapter (Chapter 7).
- **MSA Subset**:
    1. Implement every component JSR of the *subset* as specified in the following table.
    2. Comply with all the additional requirements per component JSR as specified in the relevant Additional Clarifications sections.

3. Comply with all the additional requirements as specified in the Additional Requirements chapter (Chapter 7).

Some component JSRs and their optional packages are *conditionally mandatory*. This means that an implementation claiming to be MSA compliant MUST *conditionally* implement these component JSRs, and comply with the clarifications outlined in the corresponding Additional Clarifications sections. Additional Clarifications for these component JSRs include a statement describing the condition on which the component JSR is to be a required part of the MSA Specification.

The following table specifies all the MSA and MSA Subset component JSRs and their optional packages:

Specification	Optional Packages	Comments
JSR 75 – PDA Optional Packages for the J2ME Platform Version 1.0	**File API** is a mandatory part of MSA and MSA Subset	This API allows access to the file system available on a device.
	PIM API is a mandatory part of MSA and MSA Subset	This API provides access to Personal Information Management functionality.
JSR 82 – Java APIs for Bluetooth Version 1.1	**Bluetooth API** is a conditionally mandatory part of MSA and MSA Subset	This API provides access to Bluetooth functionality.
	OBEX API is a conditionally mandatory part of MSA and MSA Subset	This API provides access to OBEX protocol, allowing the exchange of all kinds of objects such as files, pictures, calendar entries, and business cards.
JSR 118 - Mobile Information Device Profile Version 2.1	Is a mandatory part of MSA and MSA Subset	This API is a fundamental component of MSA.
JSR 135 - Mobile Media API Version 1.1	Is a mandatory part of MSA and MSA Subset	This API provides a standard way to access media capabilities such as audio and video playback.
JSR 139 - Connected Limited Device Configuration* Version 1.1	Is a mandatory part of MSA and MSA Subset	This API is a fundamental component of MSA.

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

Specification	Optional Packages	Comments
JSR 184 - Mobile 3D Graphics API for J2ME Version 1.1	Is a mandatory part of MSA and MSA Subset	This API provides access to three dimensional (3D) graphics.
JSR 205 - Wireless Messaging API 2.0 Version 2.0	Is a mandatory part of MSA and MSA Subset	This API provides access to messaging features such as SMS and MMS.
JSR 226 - Scalable 2D Vector Graphics API for J2ME Version 1.0	Is a mandatory part of MSA and MSA Subset	This API provides access to two dimensional (2D) Scalable Vector Graphics.
JSR 172 - J2ME Web Services Specification Version 1.0	**XML Parsing** is a mandatory part of MSA	This API provides a standard way to support XML parsing in a mobile device.
	Web Services (JAX RPC) is a mandatory part of MSA	This API provides access to basic web services.
JSR 177 – Security and Trust Services API for J2ME Version 1.0	**SATSA-CRYPTO** is a mandatory part of MSA	This API provides a generic way to access cryptographic services.
	SATSA-APDU is a conditionally mandatory part of MSA	This API provides a standard way to communicate with Smart Cards.
	SATSA-PKI is a conditionally mandatory part of MSA	This API provides a generic way to access Public Key Infrastructure services.
	SATSA-JCRMI is not part of MSA or MSA Subset	Not part of MSA or MSA Subset.
JSR 179 – Location API for J2ME Version 1.0.1	Is a conditionally mandatory part of MSA	This API provides a technology independent way to access location information.
JSR 180 – SIP API for J2ME Version 1.0.1	Is a mandatory part of MSA	This API provides a standard, low-level access to Session Initiation Protocol (SIP).
JSR 211 - Content Handler API Version 1.0	Is a mandatory part of MSA	This API enables the launching of Java applications based on content type.
JSR 229 - Payment API Version 1.1.0	Is a mandatory part of MSA	This API provides access to payment mechanisms.

Specification	Optional Packages	Comments
JSR 234 - Advanced Multimedia Supplements Version 1.0	Is a mandatory part of MSA	This API extends JSR 135 to provide more advanced multimedia capabilities.
JSR 238 - Mobile Internationalization API Version 1.0	Is a mandatory part of MSA	This API enables the development of localised applications.

Note: For any of the component JSRs, the version number denotes the required version of a respective configuration, profile or optional package (for example, MIDP 2.1). Backwards compatible Maintenance Releases of any component JSRs MAY be used in compliant implementations. A later version of a configuration, profile, or optional package (for example, MIDP 3.0) that includes new features MAY also be used in a compliant implementation, as long as the later version is backwards compatible with the required version in this specification, and its inclusion is clearly noted in the documentation of the device.

Backwards 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 backwards compatible.

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

6. Additional Clarifications (normative)

An implementation claiming to be MSA compliant MUST implement all component JSRs that are mandatory parts of either MSA or MSA Subset. Implementations of component 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 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 is to be a required part of the MSA Specification.

Note: An implementation compliant with the MSA Subset SHOULD NOT include any additional MSA component JSRs. When a capability is required that is not provided by the MSA Subset but is available in one of the MSA component JSRs, implementers SHOULD create a full MSA Compliant implementation. However, when business needs or external specifications dictate the implementation of the MSA Subset in conjunction with MSA component JSRs that are not part of the MSA Subset, then the implementation SHOULD include the MSA clarifications that are related to those additional MSA component JSRs.

6.1 J2ME Connected Limited Device Configuration (JSR 139)

The MSA Specification uses the J2ME Connected Limited Device Configuration version 1.1 (CLDC 1.1) as the minimum required Java ME configuration.

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.

CLDC 1.1 (JSR 139) is a superset of the earlier CLDC 1.0 (JSR 30) standard.

6.1.1 Rationale for Inclusion

CLDC is a fundamental component of the Java ME platform.

6.1.2 Condition for Inclusion

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

6.1.3 Clarifications

6.1.3.1 Minimum Heap Size

Clarification ID: CID.139.1

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

  CLDC Specification version 1.1, Section 2.2.1 “Hardware Requirements”.

  See also the “Hardware Requirements and Recommendations” chapter of this specification (especially the clarification “Java Heap Size Available to a MIDlet”).

- **Requirement Text:**

  At least 1024 kilobytes of volatile memory (such as DRAM) MUST be available for the Java virtual machine runtime (for example, the object heap). This requirement overrides the 32-kilobyte minimum volatile memory requirement defined in the CLDC Specification, version 1.1, Section 2.2.1, “Hardware requirements”.

- **Justification/Notes:**

  The 32-kilobyte minimum heap size specified by the CLDC specification, version 1.1, is too low for advanced games and media-rich applications. Without a higher minimum memory requirement, the compatibility and interoperability of applications suffers.

6.1.3.2 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.

  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 Specification.

- **Justification/Notes:**

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

6.1.3.3 Output Format of System Property microedition.platform

Clarification ID: CID.139.3

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

  CLDC Specification version 1.1, Section 6.2.10 “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, Section 6.2.10, 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.

6.1.3.4 System Property for MSA Version

Clarification ID: CID.139.4

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

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

- **Requirement Text:**

  An MSA compliant implementation MUST support the following system property (through the `System.getProperty()` method):

System Property	Explanation
`microedition.msa.version`	Version number of the supported MSA Specification. MUST be `1.0` or `1.0-SUBSET` for implementations conforming to this specification.

- **Justification/Notes:**

  Applications can query this property to learn whether an implementation is compliant with the MSA Specification, and which version of the specification is supported.

6.2 Mobile Information Device Profile (JSR 118)

JSR 118 defines the Mobile Information Device Profile version 2.1 (MIDP 2.1).

6.2.1 Rationale for Inclusion

MIDP is a fundamental component of MSA.

6.2.2 Condition for Inclusion

None.

6.2.3 Clarifications

6.2.3.1 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.

  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.

6.2.3.2 Image Object Size

Clarification ID: CID.118.2

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

  MIDP 2.1 Specification, page 16.

- **Requirement Text:**

  A compliant implementation MUST support at least the creation of `Image` objects (regardless of the format) with sizes equal to (screen width) by (screen height) by (colour depth in bits) or 262144 bits (128x128x16 bits = 32 kB), whichever value is greater. Note that the internal representation of an `Image` object SHOULD hold at least 16 bits of colour/transparency data per pixel.

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

- **Justification/Notes:**

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

6.2.3.3 Support for Communication Protocols

Clarification ID: CID.118.3

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

  MIDP 2.1 Specification, pages 55-60.

  Package `javax.microedition.io`

- **Requirement Text:**

  A compliant implementation 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.

6.2.3.4 Protocols Supported by PushRegistry

Clarification ID: CID.118.4

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

  MIDP 2.1 Specification, pages 98-101.

  Class `javax.microedition.io.PushRegistry`

- **Requirement Text:**

  In a compliant implementation the 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.

6.2.3.5 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.

- **Requirement Text:**

  A compliant implementation 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 compliant implementation must support.

6.2.3.6 MIDlet Suite 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.

- **Requirement Text:**

  A compliant implementation MUST provide the following minimum storage capacity for JAD and JAR files to support OTA download and installation of MIDlet suites:

  - At least 10 kB for a JAD file
  - At least 300 kB for a JAR file

  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 a need exists to define a minimum supported size for JAD and JAR files.

6.2.3.7 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.

- **Requirement Text:**

  A compliant implementation MUST support at least 512 attributes in both JAD files and manifests.

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

  A compliant implementation 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.

6.2.3.8 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.

- **Requirement Text:**

  A compliant implementation MUST support MIDlet suites containing any number of MIDlets between one and five (inclusive). A compliant implementation 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.

6.2.3.9 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.

- **Requirement Text:**

  A compliant implementation MUST be able to honour requests by MIDlet suites for an RMS data size of at least 64 kB. This means that if a MIDlet suite requests to reserve 64 kB (or less) for its RMS data (using its `MIDlet-Data-Size` attribute), an 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 honour the request because the device memory is occupied with other data (including `RecordStores` of other MIDlet suites).

  However, if an implementation honours 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, a compliant implementation MUST ensure the following:

- A MIDlet suite requesting the RMS data size less than or equal to 64 kB in the `MIDlet-Data-Size` attribute can be successfully installed and MIDlets from that suite can run.
- If a MIDlet from the suite 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 from the suite 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

6.2.3.10 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.

- **Requirement Text:**

  A compliant implementation 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.

6.2.3.11 Behaviour of MIDlet.platformRequest()

Clarification ID: CID.118.11

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

  MIDP 2.1 Specification, pages 443-444.

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

- **Requirement Text:**

  In a compliant implementation, method `MIDlet.platformRequest()` MUST support at least the following types of requests:

- URL pointing to a web site (`http://...` , according to RFC 2716). In this case, the implementation MUST either launch the native 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.

- URL pointing to a phone number (tel:<number>, according to RFC 3966). In this case, the implementation MUST either launch the native 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 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 behaviour in different implementations.

6.2.3.12 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.

- **Requirement Text:**

A compliant implementation MUST support the wav format (8-bit, 8 KHz, mono linear PCM wav format). A compliant implementation MUST support sampled audio.

A compliant implementation MUST support JPEG format.

- **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.

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

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

6.3.1 Rationale for Inclusion

PDA Optional Packages for the Java ME Platform contain 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).

6.3.2 Condition for Inclusion

None.

6.3.3 Clarifications

6.3.3.1 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.

6.3.3.2 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`
  - `ATTR_NONE`

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

Field	Attribute
All fields	`ATTR_NONE`
`TEL`	`ATTR_FAX, ATTR_HOME, ATTR_MOBILE, ATTR_PREFERRED, ATTR_WORK, ATTR_NONE`

- **Justification/Notes:**

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

6.3.3.3 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:**

  `ContactList` 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.

6.3.3.4 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.

6.3.3.5 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.

6.3.3.6 Order of Contact Lists Returned by PIM.listPIMLists(PIM.CONTACT_LIST)

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.

6.3.3.7 Localisation 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 a PIM API MUST be the same as in the native PIM applications. That is, if these names and labels are localised

in the native PIM applications, the PIM APIs implementation MUST also use the same localised values.

- **Justification/Notes:**

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

6.3.3.8 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.

6.3.3.9 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.

6.3.3.10 Mark/Reset Functionality in Implementations of InputStream and DataInputStream

Clarification ID: CID.75.10

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

  FileConnection Optional Package Specification.

  Class `javax.microedition.pim.PIM`
  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()”.

6.3.3.11 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 a suite's MIDlets can store its data. This directory MUST be created when a MIDlet suite is installed on the device and deleted when a MIDlet suite is removed. If a MIDlet suite is removed and the directory is not empty, an 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 2.1 rules on the inheritance of RMS record stores (pages 22-23 of the MIDP 2.1 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. Localised 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 (same as with RMS record stores). 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.

6.3.3.12 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.

6.3.3.13 Unescaping of URLs Before Security Checks

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`.

6.3.3.14 Directory Locator System Properties

Clarification ID: CID.75.14

- **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	Explanation
`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.

System Property	Explanation
`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 of MIDlet suite's storage directory.

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 a 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 pointed by a system property.

Directory names can be localised, 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 localised 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	Explanation
`fileconn.dir.photos.name`	Localised name of directory corresponding to the system property `fileconn.dir.photos.`
`fileconn.dir.videos.name`	Localised name of directory corresponding to the system property `fileconn.dir.videos.`
`fileconn.dir.graphics.name`	Localised name of directory corresponding to the system property `fileconn.dir.graphics.`
`fileconn.dir.tones.name`	Localised name of directory corresponding to the system property `fileconn.dir.tones.`
`fileconn.dir.music.name`	Localised name of directory corresponding to the system property `fileconn.dir.music.`
`fileconn.dir.recordings.name`	Localised name of directory corresponding to the system property `fileconn.dir.recordings.`

System Property	Explanation
`fileconn.dir.roots.names`	Localised names corresponding to roots returned by `FileSystemRegistry.listRoots()` method. One localised name corresponds to each root returned by the method. Localised names are in the same order as returned by the method and are separated by a semicolon (;) character. If no localised name exists for the root, the non-localised (logical) name is returned in the property for this root. Root names returned through this property cannot contain the semicolon (;) character.
`fileconn.dir.private.name`	Localised 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.

6.3.3.15 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.

6.3.3.16 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.

6.3.3.17 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.

6.3.3.18 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.

6.4 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.

6.4.1 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.

6.4.2 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.

6.4.3 Clarifications

6.4.3.1 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.

- **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.

6.4.3.2 Bluetooth Power-Save Mode

Clarification ID: CID.82.2

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

  Java APIs for Bluetooth Specification v1.1.

- **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.

6.5 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.

6.5.1 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.

6.5.2 Condition for Inclusion

None.

6.5.3 Clarifications

6.5.3.1 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:**

  A compliant 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.

6.5.3.2 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:**

  All compliant implementations MUST support the 3GPP Adaptive Multi Rate, Narrowband (AMR-NB) content format for playback of sampled audio. 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 MSA compliant implementation (this requirement does not apply to implementations supporting only MSA Subset), the `method` javax.microedition.amms.SoundSource3D.addPlayer MUST support adding `Players` 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 behaviour 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 mandates support for AMR-NB content format.

6.5.3.3 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.

6.5.3.4 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 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`.

- **Justification/Notes:**

This clarification explains certain aspects of the relationship between the behaviour of `Players` and the MIDlet Paused state.

6.5.3.5 Full-Screen Behaviour 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.

6.5.3.6 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.

6.5.3.7 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.

6.6 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)

6.6.1 Rationale for Inclusion

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

6.6.2 Condition for Inclusion

None.

6.6.3 Clarifications

6.6.3.1 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.

6.6.3.2 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.

6.7 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 utilised 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.

6.7.1 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.

6.7.2 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.

6.7.3 Clarifications

6.7.3.1 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.

6.7.3.2 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:**

  A compliant implementation MUST define the necessary Access Control mechanisms. The implementation 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.

6.7.3.3 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:**

  A compliant implementation 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.

6.7.3.4 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`.

- **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.

6.7.3.5 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`.

- **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.

6.7.3.6 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`.

- **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.

6.8 Location API for J2ME (JSR 179)

JSR 179 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.

6.8.1 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.

6.8.2 Condition for Inclusion

JSR 179 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)

6.8.3 Clarifications

No Additional Clarifications.

6.9 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.

6.9.1 Rationale for Inclusion

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

6.9.2 Condition for Inclusion

None.

6.9.3 Clarifications

6.9.3.1 SIP Methods Support

Clarification ID: CID.180.1

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

  SIP API for J2ME Specification.

  Interface `SipClientConnection`
  Interface `SipServerConnection`

- **Requirement Text:**

  All compliant implementations MUST support at least the methods defined in RFC 2976, RFC 3261, RFC 3262, RFC 3265, RFC 3311, RFC 3428, RFC 3515, and RFC 3903.

  In particular, implementations MUST support:

  - Sending INFO, REGISTER, OPTIONS, INVITE, CANCEL, BYE, ACK, PRACK, SUBSCRIBE, NOTIFY, UPDATE, MESSAGE, REFER, and PUBLISH requests on the `SipClientConnection` interface
  - Receiving INFO, OPTIONS, INVITE, CANCEL, BYE, ACK, PRACK, SUBSCRIBE, NOTIFY, UPDATE, MESSAGE, and REFER requests on the `SipServerConnection` interface

  Implementations MUST also freely allow sending and receiving of any other non-dialog-creating requests, whether in-dialog or out-of-dialog, as described in RFC3261.

The API contains dedicated methods for initiating some of these requests. Each request type is only required to be supported using the appropriate API method (as defined in the JSR 180 specification).

These RFCs have requirements for both the SIP protocol stack underneath the JSR 180 API as well as for the application level corresponding to the applications using the API. The SIP protocol stack MUST implement those requirements of the RFCs that are relevant for the stack, but following the application level requirements is the responsibility of the applications using the JSR 180 API. The responsibilities are approximately divided as follows:

Following are the requirements of the SIP protocol stack:

- Support the defined method type
- Support possible new header types defined in the RFC
- Support the responses defined in the RFC
- Manage basic transactions for new requests and responses
- Create and manage the dialog, if the request creates a dialog

Following are the requirements of the application level:

- Send appropriate requests and responses in the correct order
- Fill required user headers
- Fill required content with the required content type
- Maintain and provide event states and content for the PUBLISH method, as defined in RFC 3903
- Other application=specific requirements defined by the RFC that are not directly related to the SIP protocol

- **Justification/Notes:**

JSR 180 does not explicitly declare which SIP protocol requests are supported via the API. The API itself can support all possible requests, but it does not explicitly mandate the support for them.

6.9.3.2 Transport Protocol Support

Clarification ID: CID.180.2

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

SIP API for J2ME Specification. This clarification applies to the SIP protocol stacks supported through the JSR 180 API.

- **Requirement Text:**

  All compliant implementations MUST support sending and receiving SIP messages, at least on UDP and TCP transport protocols, as defined in RFC3261 (section 18). The default transport protocol MUST be UDP. Whenever requested, the preferred transport protocol MUST be indicated with the `;transport={transport}` parameter within the URI indicated in the `Connector.open()` method, either for client or server connections.

- **Justification/Notes:**

  The JSR 180 specification does not mandate any particular underlying transfer protocol for SIP.

6.9.3.3 SIP Authentication

Clarification ID: CID.180.3

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

  SIP API for J2ME Specification. This clarification applies to the SIP protocol stacks supported through the JSR 180 API.

- **Requirement Text:**

  The implementations MUST support at least the Digest authentication, as defined in RFC 3261.

  Implementations that interface with a GSM or IMS/MMD identity module MUST also support Digest-AKA authentication.

  Implementations MUST handle both 401 and 407 responses if the required authentication mechanism is supported.

  Implementations MUST support the invocation of `javax.microedition.sip.SipClientConnection.setCredentials` both in Initialized and Unauthorized states.

- **Justification/Notes:**

  This clarification is necessary for uniform authentication support.

6.9.3.4 SIP Dialogs

Clarification ID: CID.180.4

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

  SIP API for J2ME Specification. This clarification applies to obtaining `SipDialog` instances from `SipConnection`.

- **Requirement Text:**

  Implementations MUST support at least the creation of dialogs based on INVITE, SUBSCRIBE/NOTIFY, and REFER/NOTIFY requests, in accordance with RFC 3515.

- **Justification/Notes:**

  This clarification removes ambiguity about which requests can result in creation of `SipDialog` instances.

6.9.3.5 Refresh Support

Clarification ID: CID.180.5

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

  SIP API for J2ME Specification. This clarification applies to the SIP protocol stacks supported through the JSR 180 API.

- **Requirement Text:**

  The implementations MUST support at least the refresh of the REGISTER, SUBSCRIBE, and PUBLISH methods.

- **Justification/Notes:**

  This clarification removes ambiguity about for which SIP methods refresh needs to be supported.

6.10 Mobile 3D Graphics API (JSR 184)

JSR 184 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.

6.10.1 Rationale for Inclusion

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

6.10.2 Condition for Inclusion

None.

6.10.3 Clarifications

6.10.3.1 Security When Loading M3G Content Over Network Connections

Clarification ID: CID.184.1

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

  Mobile 3G 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 has the same requirements for 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 184 leaves open the security aspect of using a networking protocol. This requirement makes it consistent with other APIs that are part of the MSA.

6.10.3.2 Relation to the MIDlet Paused State

Clarification ID: CID.184.2

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

  Mobile 3G Graphics API. This clarification applies to implementations of the JSR 184 API and the MIDlet Paused state.

- **Requirement Text:**

  The implementation of the JSR 184 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 184 API.

- **Justification/Notes:**

  Although the MIDP 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 184 API.

6.10.3.3 JPEG IS a Mandatory Format for Image2D

Clarification ID: CID.184.3

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

  Mobile 3G Graphics API.

  Class `javax.microedition.m3g.Image2D`
  Class `javax.microedition.m3g.Loader`

- **Requirement Text:**

  JPEG (with the same detailed definitions about the JPEG image format as defined in the JSR 118 MIDP 2.1 specification for LCDUI) MUST be supported by compliant implementations as a 2D bitmap `image` format for the Image2D class using the Loader class, and for M3G content files referencing bitmap images.

  For colour JPEG images, the pixel format of the returned Image2D object MUST be Image2D.RGB and for monochrome JPEG images, the pixel format MUST be Image2D.LUMINANCE.

- **Justification/Notes:**

  JSR 184 mandates only PNG as a mandatory image format and JSR 185 mandates JPEG for LCDUI. This clarification makes support for JPEG consistent for the component JSRs of MSA.

6.10.3.4 Inconsistent Exception Condition in Loader

Clarification ID: CID.184.4

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

  Mobile 3G 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:**

  The condition for throwing `IOException` MUST be implemented as follows:

  `IOException` MUST be thrown if the data in `name/data`, or in any resource referenced from it, does not comply with file format specifications supported by the platform implementation.

- **Justification/Notes:**

  The `Loader` class description in JSR 184 specifies that:

  “Some implementations may support other formats as well. If the data is not in a supported format, is otherwise invalid, or can not be loaded for some other reason, an exception is thrown.” However, the throws clause in the `load` method description is inconsistent with this. It requires the `IOException` to be thrown if the data is not in the M3G or PNG format.

  MSA requires the JPEG bitmap image format be supported. This clarification provides an unambiguous way to support JPEG images with JSR 184.

6.11 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.

6.11.1 Rationale for Inclusion

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

6.11.2 Condition for Inclusion

None.

6.11.3 Clarifications

6.11.3.1 Handling of Messages If the Associated Java Application Isn't Running

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.

6.11.3.2 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.

6.11.3.3 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.

6.11.3.4 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:**

  In `MessagePart` objects, the `encoding` parameter maps 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`.

6.11.3.5 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 behaviour when a new message is created that doesn't match the transport protocol (SMS or MMS) specified when the message connection object was created. It also specifies exception behaviour when an attempt is made to use an inappropriate type of message for a given transport.

6.11.3.6 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 by compliant implementations as specified in Appendix D of the JSR 205 specification.

- **Justification/Notes:**

  JSR 205 does not mandate any particular underlying messaging protocols.

6.11.3.7 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, `1.0`.

- **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.

6.12 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.

6.12.1 Rationale for Inclusion

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

6.12.2 Condition for Inclusion

None.

6.12.3 Clarifications

No Additional Clarifications.

6.13 Scalable 2D Vector Graphics API for J2ME (JSR 226)

JSR 226 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.

6.13.1 Rationale for Inclusion

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

6.13.2 Condition for Inclusion

None.

6.13.3 Clarifications

6.13.3.1 System Property for Scalable 2D Vector Graphics API Version

Clarification ID: CID.226.1

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

  Scalable 2D Vector Graphics API for J2ME Specification, Chapter 1 “Overview”.

- **Requirement Text:**

  The following system property MUST be supported to discover the version of Scalable 2D Vector Graphics API supported by the device:

System Property	Explanation
`microedition.m2g.version`	Version of the Scalable 2D Vector Graphics API supported by the device (for example, `1.0`).

- **Justification/Notes:**

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

6.13.3.2 System Property for Discovering the Version and Base Profile of Supported SVG

Clarification ID: CID.226.2

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

  Scalable 2D Vector Graphics API for J2ME.

  Method `java.lang.System.getProperty(String)`

- **Requirement Text:**

  The following system properties MUST be supported to indicate the base profile and version of SVG supported by the device:

System Property	Explanation
`microedition.m2g.svg.baseProfile`	The base profile of SVG that is supported by the underlying implementation. It is `tiny` for JSR 226 v1.0 implementations.
`microedition.m2g.svg.version`	The version of SVG that is supported by the underlying implementation. For example, if the underlying SVG engine conforms to the W3C SVGT 1.1 specification, the returned string is `1.1`, and for SVGT 1.2 it is `1.2`, and so on.

- **Justification/Notes:**

  A need exists to define system properties for discovering the base profile and version of SVG supported by the device. JSR 226 does not define these system properties.

6.14 Payment API (JSR 229)

JSR 229 defines a generic API to initiate payment transactions. It defines the syntax for the description of the associated provisioning data, enabling API implementers to support different payment instruments.

Both the API and the syntax allow third-party developers to build applications with feature and service controls that are chargeable.

6.14.1 Rationale for Inclusion

This API provides access to various payment mechanisms.

6.14.2 Condition for Inclusion

None. However, support for any payment adapter is NOT REQUIRED.

6.14.3 Clarifications

No Additional Clarifications.

6.15 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.

6.15.1 Rationale for Inclusion

This API enables the creation of advanced multimedia applications.

6.15.2 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 compliant devices.

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

- All compliant implementations MUST support Image Encoding and Image Post-Processing Capabilities.
- If the device has one or more cameras, the Camera Media 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.

6.15.3 Clarifications

6.15.3.1 MediaProcessor.abort() is Synchronous

Clarification ID: CID.234.1

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

  Advanced Multimedia Supplements Specification.

  Interface `MediaProcessor`

  Method `public void abort()`

- **Requirement Text:**

  The abort method is synchronous and returns only once the processing is aborted and moved back to the `UNREALIZED` state unless the `MediaProcessor` was in the `REALIZED` state, in which case the request is ignored as specified in JSR 234.

- **Justification/Notes:**

  The JSR 234 v1.0 Specification does not clearly define whether the abort method is synchronous or asynchronous.

6.15.3.2 MediaProcessor.abort() Does Not Throw MediaException

Clarification ID: CID.234.2

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

  Advanced Multimedia Supplements Specification.

  Interface `MediaProcessor`

- **Requirement Text:**

  The class description of `MediaProcessor` in JSR 234 v1.0 contains a table describing the states and the effects of methods. This table contains the following statement for the `abort` method in the `STARTED` and `STOPPED` states:

  “On failure: Throws `MediaException`”

  This statement is incorrect because the `abort` method does not declare any exceptions. The `abort` method does not throw `MediaException`.

- **Justification/Notes:**

  This clarification corrects an incorrect statement in JSR 234 of exceptions thrown by the `abort` method (of the `MediaProcessor` class) in the `STARTED` and `STOPPED` states.

6.15.3.3 SoundSource3D.getControl Does Not Throw IllegalStateException

Clarification ID: CID.234.3

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

  Advanced Multimedia Supplements Specification.

  Interface `SoundSource3D`

  Method `public Control getControl(java.lang.String controlType)`
  Method `public Control getControls()`

- **Requirement Text:**

  The `SoundSource3D` interface methods `getControl(java.lang.String controlType)` and `getControls()` MUST NOT ever throw `IllegalStateException`.

- **Justification/Notes:**

  This clarification removes ambiguity regarding the exact exceptions that might be thrown by `SoundSource3D` interface methods.

6.15.3.4 SnapShotControl and Burst Mode Shooting

Clarification ID: CID.234.4

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

  Advanced Multimedia Supplements Specification.

  Interface `javax.microedition.amms.control.camera.SnapshotControl`

  Method `public void setDirectory(java.lang.String directory)`
  Method `public void setFilePrefix(java.lang.String prefix)`
  Method `public void setFileSuffix(java.lang.String suffix)`
  Method `public java.lang.String getDirectory()`
  Method `public java.lang.String getFilePrefix()`
  Method `public java.lang.String getFileSuffix()`

- **Requirement Text:**

  The implementation of `SnapshotControl.start()` MUST not change the storing directory, file name prefix, or filename suffix during a burst shooting session even if `setDirectory()`, `setFilePrefix()`, or `setFileSuffix()` are called during that burst shooting session. Settings caused by calls to methods `setDirectory()`, `setFilePrefix()`, or `setFileSuffix()` MUST become effective when the next burst shooting session is started with a call to start the method, but not yet during the ongoing burst session. However, `getDirectory()`, `getFilePrefix()`, and `getFileSuffix()` MUST return the new values right after the call of the `setDirectory()`, `setFilePrefix()`, or `setFileSuffix()` methods, respectively.

- **Justification/Notes:**

  This clarification removes ambiguity about how `setDirectory()`, `setFilePrefix()`, or `setFileSuffix()` behave if they are called during an ongoing burst shooting session.

6.15.3.5 FocusControl Next and Previous Methods During Autofocus

Clarification ID: CID.234.5

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

  Advanced Multimedia Supplements Specification.

  Interface `javax.microedition.amms.control.camera.FocusControl`

  Method `public int setFocus(int distance)`

- **Requirement Text:**

  If a camera is in auto focus mode and `setFocus` is called with parameter `NEXT` or `PREVIOUS` and manual focus is supported, the camera MUST switch to the manual focusing mode and move one step from the current (auto-focused) distance in a given direction. However, if the focusing distance is already the minimum returned by `getMinFocus` while calling `setFocus(PREVIOUS)` or `Integer.MAX_VALUE` while calling `setFocus(NEXT)`, only the switch to manual focusing mode MUST happen and the focusing distance MUST NOT change.

- **Justification/Notes:**

  This clarification removes ambiguity about the behaviour of `setFocus(NEXT)` or `setFocus(PREVIOUS)` while in auto focus mode.

6.16 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. This 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.

6.16.1 Rationale for Inclusion

This small but beneficial feature allows truly global application development.

6.16.2 Condition for Inclusion

None.

6.16.3 Clarifications

No Additional Clarifications.

7. Additional Requirements (normative)

The following sections contain additional requirements for MSA compliancy.

7.1 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 backwards compatibility, the JTWI requirements are included in the MSA Specification, and a compliant implementation MUST provide support for those requirements.

7.1.1 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.

7.1.2 CLDC-Related JTWI Clarifications

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

7.1.2.1 Minimum Clock Resolution

Clarification ID: CID.185.1

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

  JTWI 1.0 Specification, Section 3.3

  CLDC 1.1 Specification
  Class `java.lang.System:`
  `public long currentTimeMillis()`

- **Requirement Text:**

  In a compliant device, 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:**

  A compliant implementation MAY support a smaller increment.

7.1.2.2 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 Specification
  Class `java.util.TimeZone:`
  `public String getID()`
  `public static TimeZone getTimeZone(String ID)`

- **Requirement Text:**

  A compliant 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.

7.1.2.3 Names for Encodings

Clarification ID: CID.185.3

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

  JTWI 1.0 Specification, Section 3.5

  CLDC 1.1 Specification, Section 6.2.9
  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 backwards compatibility.

7.1.2.4 Character Properties

Clarification ID: CID.185.4

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

  JTWI 1.0 Specification, Section 3.6

  CLDC 1.1 Specification, Section 6.2.1
  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:**

  Compliant 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.

7.1.3 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.

7.1.3.1 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.
  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:**

  A compliant 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.

7.1.3.2 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.
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:**

  A compliant implementation 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.

7.1.3.3 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.
  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:**

  A compliant device MUST support the loading of 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, a compliant implementation 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.

7.1.3.4 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.
  Class `javax.microedition.lcdui.TextField`
  Class `javax.microedition.lcdui.TextBox`

- **Requirement Text:**

  Compliant devices MUST implement 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.

7.1.3.5 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.
  Class `javax.microedition.io.PushRegistry`

- **Requirement Text:**

  A compliant implementation MUST implement 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.

7.1.4 WMA-Related JTWI Clarifications

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

7.1.4.1 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 1.1 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.

7.1.4.2 Cell Broadcast in GSM Devices

Clarification ID: CID.185.11

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

  JTWI 1.0 Specification, Section 5.3

  WMA 1.1 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.

7.1.4.3 SMS Push

Clarification ID: CID.185.12

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

  JTWI 1.0 Specification, Section 5.4

  WMA 1.1 Specification, Appendix A
  MIDP 2.1 Specification, Chapter 7 “Package javax.microedition.io”, pages 98-108.
  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 D 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.

7.1.5 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.

7.1.5.1 HTTP 1.1 Protocol

Clarification ID: CID.185.13

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

  JTWI 1.0 Specification, Section 6.2

  MMAPI 1.1 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.

7.1.5.2 JPEG Encoding in Video Snapshots

Clarification ID: CID.185.14

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

  JTWI 1.0 Specification, Section 6.5

  MMAPI 1.1 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.

7.1.5.3 Tone Sequence File Format

Clarification ID: CID.185.15

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

  JTWI 1.0 Specification, Section 6.6

  MMAPI 1.1 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.

7.1.6 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 summarises 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”

7.2 Hardware Requirements

This chapter contains hardware requirements and recommendations for compliant devices. 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 devices.

7.2.1 Screen Size

Clarification ID: CID.248.1

Minimum screen resolution available to a Java application on a compliant device SHOULD be no less than 128x128 pixels. The screen resolution is returned by `Canvas.getHeight()` and `Canvas.getWidth()` in full-screen mode.

7.2.2 Colour Depth

Clarification ID: CID.248.2

Minimum colour depth (as returned by `Display.numColors()`) available to a Java application on a compliant device SHOULD be no less than 65536 colours (16 bits).

7.2.3 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 1024 kB. Further, under normal conditions, a compliant implementation SHOULD ensure that Java heap size available to a MIDlet is at least 2048 kB.

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 fifteen (15) byte arrays, of size 64 kB each. (Note: The number of byte arrays is intentionally 15 rather than 16, because 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:

```
byte[][] container = new byte[15][];

for (int i = 0; i < 15; i++) {
    byte[] barray = new byte[65536];
    container[i] = barray;
}
```

7.3 Security Requirements

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

The security requirements are normative, and therefore all compliant devices MUST adhere to the requirements in this section.

This section supersedes the Recommended Security Policy in MIDP 2.1 and JTWI (JSR 185). It requires the Recommended Security Policy of MIDP 2.1. This means that compliant devices 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 devices as well.

MIDP 2.1 defines the framework for authenticating the source of a MIDlet suite. It authorizes the MIDlet suite to perform protected functions by granting permissions the MIDlet suite has requested based on the security policy on the device. It also identifies functions that are deemed sensitive, and defines permissions for those protected functions. Additionally, MIDP 2.1 specifies common rules for APIs that can be used together with MIDP, but are specified outside MIDP. The MIDP 2.1 specification does not mandate a single trust model, but allows the model to follow the trust policy of a device.

7.3.1 General

Compliant devices implementing this Recommended Security Policy MUST follow the security framework specified in the MIDP 2.1 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.

7.3.2 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.

7.3.2.1 Mapping Permissions to Function Groups in Protected Domains

The display on a device might not be large enough to present all permissions to the user 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** – Represents permissions to any function that results in a voice call.
- **Call Control** – Represents permissions to any function that results in a call setup or teardown of a restricted network connection.
- **Net Access** – Represents permissions to any function that results in an active network data connection (for example, GSM, GPRS, UMTS). Such functions MUST be mapped to this group.
- **Low-Level Net Access** – Represents permissions to any function that results in an active low-level network data connection (for example, Sockets). Such functions MUST be mapped to this group.
- **Messaging** – Represents permissions to any function that allows sending or receiving messages (for example, SMS, MMS).
- **Restricted Messaging** – Represents permissions to any function that allows sending or receiving messages to restricted messaging service (for example, Cell Broadcast).
- **Application Auto Invocation** – Represents permissions to any function that allows a MIDlet suite to be invoked automatically (for example, push, timed MIDlets).
- **Local Connectivity** – Represents permissions to any function that activates a local port for further connection (for example, COMM port, IrDa, Bluetooth).
- **Authentication** - Represents permissions to any function that gives a MIDlet suite access to authentication functionality.

User privacy-related groups:

- **Multimedia Recording** – Represents permissions to any function that gives a MIDlet suite the ability to do any kind of multimedia recording (for example, capture still images, record video or audio clips).
- **Read User Data Access** – 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** – 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** – Represents permissions to any function that gives a MIDlet suite the ability to communicate with the smart card.
- **Location** – Represents permissions to any function that gives a MIDlet suite access to Location information.
- **Landmark** - Represents permissions to any function that gives a MIDlet suite access to Landmark information.

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 are 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.

If a new function group is to be added, the following SHOULD be taken into consideration: the group to be added MUST NOT introduce any redundancy to the existing groups and the new group MUST be capable of protecting a wide range of similar features. The latter requirement is to prevent introducing narrowly scoped groups. The new function group SHOULD be sufficiently future-proof to contain new features added by future APIs and SHOULD NOT only concern the features being initially included in it.

It is the function groups and not the individual permissions that SHOULD be presented when the user is prompted. In addition, the function groups SHOULD be presented to the user while configuring the settings for a given MIDlet suite.

Tables 1 and 2 are duplicated from the Recommended Security Policy Addendum in MIDP 2.1. These tables are included here so that all relevant MSA Security Policy information is in one document.

Table 1 presents the policy that is RECOMMENDED by the MSA specification. This policy relies on the security framework defined in MIDP 2.1. The table specifies the available permission settings for each function group defined. Settings that are effective at the time the MIDlet suite is invoked for the first time, and remain effective until the user changes them in the MIDlet suite's configuration menu, are called "default settings." Settings available to the user in the configuration menu, to which the user can change from a default setting, are called "other settings." Together, default and other settings form a pool of available configuration settings for the MIDlet suite. Default and other settings are presented for each function group and both Third Party Protection Domains. The naming of the function groups is implementation specific, but MUST follow the guidelines of the function group names defined in this document, as well as the definitions of these groups.

Tables 2 through 12 present individual permissions defined in the MIDP 2.1 specification (and other MSA and MSA Subset component JSRs) and map to the function groups specified in this section. An individual permission MUST occur in only one function group.

A compliant device SHOULD adhere to Table 1 in this specification. A compliant device MUST adhere to tables 2 through 12 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	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Restricted Messaging	default setting	Oneshot	default setting	Oneshot
	other settings	Blanket, Session, No	other settings	No
Application Auto Invocation	default setting	Oneshot	default setting	Oneshot
	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

The device MAY enhance and simplify the user experience by applying a single set of configuration settings (default or other), to not just a single MIDlet suite, but to all MIDlet suites for a given signer. This option MUST NOT compromise the function groups and

available settings defined in Table 1. If such an option exists, the user MUST be prompted to save the settings and reuse them in the future for MIDlet suites from the same signer. Such a feature MAY also inform the user that a given source has already been accepted and has an alias to the saved configuration settings. For each application, the implementation MAY read requested permissions from the `MIDlet-Permissions` and `MIDlet-PermissionsOpt` attributes, notify the user which capability the application requires, and prompt the user to accept or reject installation of the application. Implementations that support this feature MUST inform the user during application installation that the security settings for the given signer have previously been saved on the device and will be applied for the newly downloaded application.

Blanket permission given for some combinations of Function groups can lead to higher risks for the user. For MIDlet suites in the Identified Third Party Protection Domain, the user MUST be notified of the higher risk involved, and must acknowledge that this risk is accepted, to permit such combinations to be set. The combination of Blanket permission in Function groups where this applies is any of:

- Net Access
- Low Level Net Access
- Messaging
- Restricted Messaging
- Call Control
- Local Connectivity

set to Blanket, in combination with any of

- Multimedia recording
- Read User Data Access

set to Blanket.

This restriction does not apply to the Unidentified Third Party Protection Domain, because these combinations are forbidden in this domain according to Table 1.

Additionally, the Blanket setting for Application Auto Invocation and the Blanket setting for any of:

- Net Access
- Low Level Net Access

are mutually exclusive. This constraint prevents a MIDlet suite from auto-invoking itself, then accessing a chargeable network without the user's knowledge. If the user attempts to set either the Application Auto Invocation or the Net Access (or Low Level Net Access) group to Blanket when the other Function group is already in Blanket mode, the user MUST be prompted as to which of the two Function groups shall be granted Blanket and which Function group shall be granted Session.

For each Phone Call and Messaging prompt, the implementation MUST present the user with the destination phone number or a corresponding name before the user approves the action. For the Messaging group, if the implementation maps a single API call to more than one message (that is, the implementation supports disassembly and reassembly), the implementation MUST present the user with the number of messages that will actually be sent. This requirement ensures that the user always understands the network costs associated with running the program no matter what API calls are involved.

When No is selected for the function group permission, the implementation MUST behave as follows:

- During application installation, the function group permission is treated as a user permission. In particular, the application installation MUST NOT fail due to this setting being used.
- During application execution, the implementation MUST NOT present user prompts when the application tries to access the protected APIs from the function group, and such an API call MUST result in a `SecurityException`.

With user prompts in Blanket and Session interaction modes, the implementation MUST give the user a choice to deny the permission. If the user denies the permission, the implementation MUST remember this answer and MUST NOT present further user prompts until the application is uninstalled (in Blanket mode) or terminated (in Session mode).

Table 2: Assigning Permissions Specified in MIDP 2.1 to Function Groups. This table is applicable to MSA and the MSA Subset.

Permission	Protocol	Function group
`javax.microedition.io.Connector.http`	HTTP	Net Access
`javax.microedition.io.Connector.https`	HTTPs	Net Access
`javax.microedition.io.Connector.datagram`	Datagram	Low Level Net Access
`javax.microedition.io.Connector.datagramreceiver`	Datagram server (without host)	Low Level Net Access
`javax.microedition.io.Connector.socket`	Socket	Low Level Net Access
`javax.microedition.io.Connector.serversocket`	server socket (without host)	Low Level Net Access
`javax.microedition.io.Connector.ssl`	SSL	Low LevelNet Access
`javax.microedition.io.Connector.comm`	comm	Local Connectivity
`javax.microedition.io.PushRegistry`	All	Application Auto Invocation

Network Access Requirements

Unidentified third-party applications MUST use the normal `HttpConnection` and `HttpsConnection` APIs to access web and secure web services. No restrictions apply to web server port numbers through these interfaces. The implementations augment the protocol so that web servers can identify unidentified third-party applications. The Product-Token `UNTRUSTED` refers to the Unidentified Third Party Protection Domain and is retained to maintain backward compatibility with JTWI. The following MUST be implemented:

- `HttpConnection` and `HttpsConnection` MUST include in the User-Agent header with the Product-Token "UNTRUSTED/1.0". User-Agent headers supplied by the application MUST NOT be deleted.
- `SocketConnection()` using TCP sockets MUST throw `java.lang.SecurityException` when a MIDlet suite belonging to the Unidentified Third Party Protection Domain attempts to connect on ports 80,8080 (http), and 443 (https).
- `SecureConnection()` using TCP sockets MUST throw `java.lang.SecurityException` when a MIDlet suite belonging to the Unidentified Third Party Protection Domain attempts to connect on port 443 (https).
- `DatagramConnection.send()` MUST throw `java.lang.SecurityException` when a MIDlet suite belonging to the Unidentified Third Party Protection Domain attempts to send datagrams to any of the ports 9200 through 9203 (WAP Gateway).
- These requirements SHOULD be applied regardless of the API used to access the network. For example, the `javax.microedition.io.Connector.open()` and `javax.microedition.media.Manager.createPlayer()` methods SHOULD throw `java.lang.SecurityException` if access is attempted to these port numbers through a means other than the normal `HttpConnection` and `HttpsConnection` APIs.

Table 3: Assigning Permissions and API Calls Specified in the Personal Information Management Package (of the PDA Profile to Function Groups. This table is applicable to MSA and the MSA Subset.

Permission	Permitted Java API Calls	Function Group

`javax.microedition.pim.ContactList.read`	`openPIMList(PIM.CONTACT_LIST,PIM.READ_ONLY)`
	`openPIMList(PIM.CONTACT_LIST,PIM.READ_WRITE)`
	`openPIMList(PIM.CONTACT_LIST, PIM.READ_ONLY, java.lang.String)`
	`openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE, java.lang.String)`
	`listPIMLists(PIM.CONTACT_LIST)`
		Read User Data Access

Permission	Permitted Java API Calls	Function Group
javax.microedition.pim.ContactList.write	openPIMList(PIM.CONTACT_LIST,PIM.WRITE_ONLY) openPIMList(PIM.CONTACT_LIST,PIM.READ_WRITE) openPIMList(PIM.CONTACT_LIST,PIM.WRITE_ONLY, java.lang.String) openPIMList(PIM.CONTACT_LIST,PIM.READ_WRITE, java.lang.String)	Write User Data Access
javax.microedition.pim.EventList.read	openPIMList(PIM.EVENT_LIST, PIM.READ_ONLY) openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE) openPIMList(PIM.EVENT_LIST, PIM.READ_ONLY, java.lang.String) openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE, java.lang.String) listPIMLists(PIM.EVENT_LIST)	Read User Data Access
javax.microedition.pim.EventList.write	openPIMList(PIM.EVENT_LIST, PIM.WRITE_ONLY) openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE) openPIMList(PIM.EVENT_LIST, PIM.WRITE_ONLY, java.lang.String) openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE, java.lang.String)	Write User Data Access
javax.microedition.pim.ToDoList.read	openPIMList(PIM.TODO_LIST, PIM.READ_ONLY) openPIMList(PIM.TODO_LIST, PIM.READ_WRITE) openPIMList(PIM.TODO_LIST, PIM.READ_ONLY, java.lang.String) openPIMList(PIM.TODO_LIST, PIM.READ_WRITE, java.lang.String) listPIMLists(PIM.TODO_LIST)	Read User Data Access
javax.microedition.pim.ToDoList.write	openPIMList(PIM.TODO_LIST, PIM.WRITE_ONLY) openPIMList(PIM.TODO_LIST, PIM.READ_WRITE) openPIMList(PIM.TODO_LIST, PIM.WRITE_ONLY, java.lang.String) openPIMList(PIM.TODO_LIST, PIM.READ_WRITE, java.lang.String)	Write User Data Access

Permission	Permitted Java API Calls	Function Group
javax.microedition.io.Connector.file.read	javax.microedition.io.Connector open("file://...") open("file://...", Connector.READ) open("file://...", Connector.READ_WRITE) openDataInputStream("file://...") openInputStream("file://...") javax.microedition.io.file.FileConnection setFileConnection, when instance opened with READ setFileConnection, when instance opened with READ_WRITE javax.microedition.io.file.FileSystemRegistry addFileSystemListener listRoots	Read User Data Access
javax.microedition.io.Connector.file.write	javax.microedition.io.Connector open("file://...") open("file://...", Connector.WRITE) open("file://...", Connector.READ_WRITE) openDataOutputStream("file://...") openOutputStream("file://...") javax.microedition.io.file.FileConnection setFileConnection, when instance opened with WRITE setFileConnection, when instance opened with READ_WRITE	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.

Table 4: Assigning Proposed Permissions and API Calls Specified in the Bluetooth API to Function Groups. This table is applicable to MSA and the MSA Subset.

Permission	Permitted API Calls	Function Group
javax.microedition.io.Connector.bluetooth.client	Connector.open(“btspp://<server BD_ADDR>…”) Connector.open(“btl2cap://<server BD_ADDR>…”)	Local Connectivity

Permission	Permitted API Calls	Function Group
javax.microedition.io.Connector.obex.client	Connector.open("btgoep://<server BD_ADDR>…") Connector.open("irdaobex://discover…")	Local Connectivity
javax.microedition.io.Connector.obex.client.tcp	Connector.open("tcpobex://<server IP_ADDR>…")	Net Access
javax.microedition.io.Connector.bluetooth.server	Connector.open("btspp://localhost:…") Connector.open("btl2cap://localhost:…")	Local Connectivity
javax.microedition.io.Connector.obex.server	Connector.open("btgoep://localhost:…") Connector.open("irdaobex://localhost:…")	Local Connectivity
javax.microedition.io.Connector.obex.server.tcp	Connector.open("tcpobex://:<PORT>") Connector.open("tcpobex://")	Net Access

Table 5: Assigning Permissions and API Calls Specified in the Mobile Media API to Function Groups. This table is applicable to MSA and the MSA Subset.

Permission	Permitted API Calls	Function Group
javax.microedition.media.control.RecordControl	RecordControl.setRecordLocation() RecordControl.setRecordStream()	Multimedia recording
javax.microedition.media.control.VideoControl.getSnapshot	VideoControl.getSnapshot()	Multimedia recording

Table 6: Assigning Permissions and API Calls Specified in the Security and Trust Services API to Function Groups. This table is applicable only to MSA.

Permission	Permitted API Calls	Function Group
javax.microedition.apdu.sat	Connector.open("apdu:"[<slot>]"; target=SAT");	
javax.microedition.apdu.aid	Connector.open("apdu:"[<slot>]"; target="<AID>);	Smart Card Communication
javax.microedition.jcrmi	Connector.open("jcrmi:"[<slot>]"; aid="<AID>);	Smart Card Communication
javax.microedition.securityservice.CMSMessageSignatureService	CMSMessageSignatureService.authenticate (byte[] byteArrayToAuthenticate, ...)	Authentication

Permission	Permitted API Calls	Function Group
`javax.microedition.apdu.sat` permission is not assigned to any function group. Access is *Allowed* for manufacturer and operator domain, and *Denied* for other domains.		

Table 7: Assigning Permissions and API calls Specified in the Location API to Function Groups. This table is applicable only to MSA.

Permission	Permitted API calls	Function Group
`javax.microedition.location.Location`	`LocationProvider.getLocation()` `LocationProvider.setLocationListener()`	Location
`javax.microedition.location.Orientation`	`Orientation.getOrientation()`	Location
`javax.microedition.location.ProximityListener`	`LocationProvider.addProximityListener()`	Location
`javax.microedition.location.LandmarkStore.read`	`LandmarkStore.getInstance()` `LandmarkStore.listLandmarkStores()`	Landmark

`javax.microedition.location.LandmarkStore.write`	`LandmarkStore.addLandmark()`
	`LandmarkStore.deleteLandmark()`
	`LandmarkStore.removeLandmarkFromCategory()`
	`LandmarkStore.updateLandmark()`
		Landmark

`javax.microedition.location.LandmarkStore.category`	`LandmarkStore.addCategory()` `LandmarkStore.deleteCategory()`	Landmark
`javax.microedition.location.LandmarkStore.management`	`LandmarkStore.createLandmarkStore()` `LandmarkStore.deleteLandmarkStore()`	Landmark

Table 8: Assigning Permissions and API Calls Specified in the SIP API to Function Groups. This table is applicable only to MSA.

Permission	Permitted API Calls	Function Group
`javax.microedition.io.Connector.sip`	`Connector.open(<SIP URI>)`	Call Control
`javax.microedition.io.Connector.sips`	`Connector.open(<SIPS URI>)`	Call Control

Table 9: Assigning permissions and API calls specified in the Wireless Messaging API (JSR 205) to function groups (applicable to both MSA and MSA Subset). This table is applicable to MSA and the MSA Subset.

Permission	Protocol	Function group
`javax.wireless.messaging.sms.send`	SMS	Messaging
`javax.wireless.messaging.sms.receive`	SMS	Messaging
`javax.microedition.io.Connector.sms`	SMS	Messaging
`javax.microedition.io.Connector.cbs`	CBS	Restricted Messaging
`javax.microedition.io.Connector.mms`	MMS	Messaging
`javax.wireless.messaging.cbs.receive`	CBS	Restricted Messaging
`javax.wireless.messaging.mms.send`	MMS	Messaging
`javax.wireless.messaging.mms.receive`	MMS	Messaging

Interaction Modes

If the interaction mode is Oneshot for the Messaging or Restricted Messaging function group, the connections are assigned the Blanket mode as shown in TABLE 9-A.

Table 9-A: Interaction Modes for Connections.

Permission	Function Group Interaction Mode	Permission Interaction Mode
`javax.microedition.io.Connector.sms`	Oneshot	Blanket
`javax.microedition.io.Connector.cbs`	Oneshot	Blanket
`javax.microedition.io.Connector.mms`	Oneshot	Blanket

If the interaction mode is Oneshot for the Messaging or Restricted Messaging function group, the mode for the permissions for receiving messages is Blanket, as shown in TABLE 9-B. Also, the permissions for sending messages are the same as shown in TABLE 9-B. Thus, the interaction mode is Oneshot for each message sent by the MIDlet suite. When the interaction mode is No, it applies to all the individual permissions within these Messaging function groups. These requirements apply regardless of the intended recipient, whether it is a user or another application.

Table 9-B: Interaction Modes for Connections.

Permission	Function Group Interaction Mode	Permission Interaction Mode
`javax.wireless.messaging.sms.send`	Oneshot	Oneshot

Permission	Function Group Interaction Mode	Permission Interaction Mode
javax.wireless.messaging.sms.receive	Oneshot	Blanket
javax.wireless.messaging.cbs.receive	Oneshot	Blanket
javax.wireless.messaging.mms.send	Oneshot	Oneshot
javax.wireless.messaging.mms.receive	Oneshot	Blanket

Table 10: Assigning Permissions and API Calls Specified in the Content Handler API to Function Groups This table is applicable only to MSA.

Permission	Permitted API Calls	Function Group
javax.microedition.content.ContentHandler	Registry.register()	Application Auto Invocation

Table 11: Assigning Permissions and API Calls Specified in the Payment API to Function Groups. This table is applicable only to MSA.

Permission	Permitted API Calls	Function Group
javax.microedition.payment.process	TransactionModule.process()	
This permission is not assigned to any function group. Access is *Denied* under Unidentified Third Party Domain and *Allowed* under Manufacturer, Operator, and Identified Third Party domains.		

Table 12: Assigning Permissions and API Calls Specified in the Advanced Multimedia Supplements API to Function Groups. This table is applicable only to MSA.

Permission	Permitted API Calls	Function Group
javax.microedition.amms.control.camera.enableShutterFeedback	CameraControl.enableShutterFeedback()	Multimedia Recording
javax.microedition.amms.control.tuner.setPreset	Tuner.setPreset(int preset) Tuner.setPreset(int preset, int freq, java.lang.String mod, int stereomode)	Write User Data Access

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.

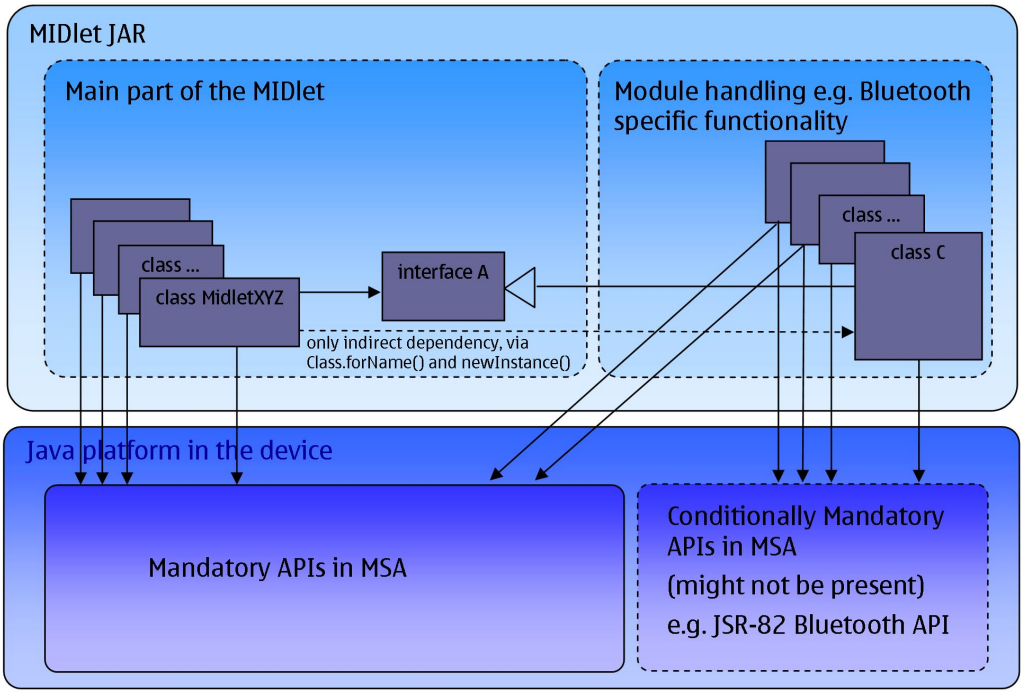


Figure 1. Dependencies between the classes of the MIDlet and the platform

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 both MSA and MSA Advanced is to produce well defined, evolving Java platforms that meet the latest market requirements and the anticipated speed of technology evolution.

The timing of the future MSA and MSA Advanced 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 backwards 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 (and its version):

Property Name	Defined in JSR	Present in
`microedition.io.file.FileConnection.version`	75	MSA and MSA Subset
`microedition.pim.version`	75	MSA and MSA Subset
`bluetooth.api.version`	82	MSA and MSA Subset
`obex.api.version`	82	MSA and MSA Subset
`microedition.profiles`	118, 139 and 248	MSA and MSA Subset
`microedition.media.version`	135	MSA and MSA Subset
`microedition.configuration`	139	MSA and MSA Subset
`microedition.m3g.version`	184	MSA and MSA Subset
`wireless.messaging.version`	205 and 248	MSA and MSA Subset
`microedition.m2g.version`	226 and 248	MSA and MSA Subset
`microedition.msa.version`	248	MSA and MSA Subset
`xml.jaxp.subset.version`	172 and 248	MSA
`xml.rpc.subset.version`	172 and 248	MSA
`microedition.satsa.apdu.version`	177 and 248	MSA
`microedition.satsa.crypto.version`	177 and 248	MSA
`microedition.satsa.pki.version`	177 and 248	MSA
`microedition.location.version`	179	MSA
`microedition.sip.version`	180	MSA
`microedition.chapi.version`	211	MSA
`microedition.payment.version`	229 and 248	MSA
`microedition.amms.version`	234	MSA
`microedition.global.version`	238	MSA

A.1.2 Other System Properties

The following table summarizes all other properties specified in the MSA component JSRs or in the additional clarifications:

Property Name	Defined in JSR	Present in
file.separator	75	MSA and MSA Subset
fileconn.dir.photos	248	MSA and MSA Subset
fileconn.dir.videos	248	MSA and MSA Subset
fileconn.dir.graphics	248	MSA and MSA Subset
fileconn.dir.tones	248	MSA and MSA Subset
fileconn.dir.music	248	MSA and MSA Subset
fileconn.dir.recordings	248	MSA and MSA Subset
fileconn.dir.private	248	MSA and MSA Subset
fileconn.dir.photos.name	248	MSA and MSA Subset
fileconn.dir.videos.name	248	MSA and MSA Subset
fileconn.dir.graphics.name	248	MSA and MSA Subset
fileconn.dir.tones.name	248	MSA and MSA Subset
fileconn.dir.music.name	248	MSA and MSA Subset
fileconn.dir.recordings.name	248	MSA and MSA Subset
fileconn.dir.roots.name	248	MSA and MSA Subset
fileconn.dir.private.name	248	MSA and MSA Subset
bluetooth.l2cap.receiveMTU.max	82	MSA and MSA Subset
bluetooth.connected.devices.max	82	MSA and MSA Subset
bluetooth.connected.inquiry	82	MSA and MSA Subset
bluetooth.connected.page	82	MSA and MSA Subset
bluetooth.connected.inquiry.scan	82	MSA and MSA Subset
bluetooth.connected.page.scan	82	MSA and MSA Subset
bluetooth.master.switch	82	MSA and MSA Subset
bluetooth.sd.trans.max	82	MSA and MSA Subset
bluetooth.sd.attr.retrievable.max	82	MSA and MSA Subset
microedition.locale	118	MSA and MSA Subset
microedition.commports	118 and 248	MSA and MSA Subset
microedition.hostname	118	MSA and MSA Subset
supports.mixing	135	MSA and MSA Subset
supports.audio.capture	135	MSA and MSA Subset
supports.video.capture	135	MSA and MSA Subset
supports.recording	135	MSA and MSA Subset
audio.encodings	135	MSA and MSA Subset
video.encodings	135	MSA and MSA Subset
video.snapshot.encodings	135	MSA and MSA Subset
streamable.contents	135	MSA and MSA Subset

Property Name	Defined in JSR	Present in
microedition.platform	139 and 248	MSA and MSA Subset
microedition.encoding	139	MSA and MSA Subset
wireless.messaging.sms.smsc	205	MSA and MSA Subset
wireless.messaging.mms.mmsc	205	MSA and MSA Subset
microedition.m2g.svg.baseProfile	226 and 248	MSA and MSA Subset
microedition.m2g.svg.version	226 and 248	MSA and MSA Subset
microedition.smartcardslots	177	MSA
supports.mediacapabilities	234	MSA
tuner.modulations	234	MSA
audio.samplerates	234	MSA
audio3d.simultaneouslocations	234	MSA
camera.orientations	234	MSA
camera.resolutions	234	MSA

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 compliant implementations. The tables distinguish between items that might be mandatory for full MSA or for the MSA Subset.

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 compliant devices, and whether the protocol is mandatory or optional. Refer to the legend below the table.

Protocol	Java ME API	JSR	MSA	MSA Subset
HTTP client	`javax.microedition.io.HttpConnection`	118	M	M
HTTP over SSLv3.0 (HTTPS) client	`javax.microedition.io.HttpsConnection`	118	M*	M*
HTTP over TLS v1.0 (HTTPS) client	`javax.microedition.io.HttpsConnection`	118	O	O
TCP socket client	`javax.microedition.io.SocketConnection`	118	M	M
TCP socket server	`javax.microedition.io.ServerSocketConnection`	118	O	O
TLS v1.0 client	`javax.microedition.io.SecureConnection`	118	O	O
SSL v3.0 client	`javax.microedition.io.SecureConnection`	118	M*	M*
UDP	`javax.microedition.io.UDPDatagramConnection`	118	O	O
SIP	`javax.microedition.sip.SIPConnection`	180	M	-
SMS	`javax.wireless.messaging.MessageConnection`	120/ 205	M	M

* An implementation that supports TLS v1.0 and supports section E of RFC 2246 (to provide backwards compatibility with SSL v3.0 servers) is compliant with this requirement.

Protocol	Java ME API	JSR	MSA	MSA Subset
MMS	`javax.wireless.messaging.MessageConnection`	205	M	M
CBS	`javax.wireless.messaging.MessageConnection`	120/205	O	O
Bluetooth L2CAP client	`javax.bluetooth.L2CAPConnection`	82	C	C
Bluetooth L2CAP server	`javax.bluetooth.L2CAPConnectionNotifier`	82	C	C
Bluetooth RFCOMM client	`javax.microedition.io.StreamConnection`	82	C	C
Bluetooth RFCOMM server	`javax.microedition.io.StreamConnectionNotifier`	82	C	C
Bluetooth OBEX client	`javax.obex.ClientSession`	82	C	C
Bluetooth OBEX server	`javax.obex.SessionNotifier`	82	C	C
IrDA OBEX client	`javax.obex.ClientSession`	82	C	C
IrDA OBEX server	`javax.obex.SessionNotifier`	82	C	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 compliant devices and whether the format is mandatory or optional. Refer to the legend below the table.

Content format	Java API	JSR	MSA	MSA Subset
PNG	`javax.microedition.lcdui.Image`	118	M	M
PNG	`javax.microedition.m3g.Image2D`	184	M	M
JPEG (JFIF)	`javax.microedition.lcdui.Image`	118	M	M
JPEG (JFIF)	`javax.microedition.m3g.Image2D`	184	M	M
M3G	`javax.microedition.m3g.Object3D`	184	M	M
SVG Tiny 1.1	`javax.microedition.m2g.SVGImage`	226	M	M
8 kHz, 8-bit linear PCM audio in WAV	`javax.microedition.media.Player`	135	M	M
AMR-NB*	`javax.microedition.media.Player`	135	M	M
MIDI	`javax.microedition.media.Player`	135	M	M
SP-MIDI	`javax.microedition.media.Player`	135	M	M

Notation: **M** – Mandatory

* 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	Requirement	Recommendation
Minimum supported screen size	None	128 by 128
Minimum supported number of colours	None	65536 16 bits
Pixel aspect ratio	None	1:1
Minimum heap size available to MIDlet	1024 kB	2048 kB
Minimum supported clock resolution	None	40 ms
Minimum supported number of application-created threads per MIDlet	10	None
Minimum supported MIDlet JAR file download size	300kB	None
Minimum supported MIDlet JAR file install size	300kB	None
Minimum supported MIDlet suite JAD size	10 kB	None
Minimum supported size of attribute values in JAD manifest	2048 bytes	None
Minimum supported size of attribute name	70 bytes	None
Minimum supported number of attributes in JAD manifest	512	None
Minimum supported number of MIDlets in a MIDlet suite	5	None
Minimum supported RMS data size per MIDlet suite	64 kB	None
Minimum supported number of independent record stores per MIDlet suite	10	None
Minimum supported image object size	Size of full screen full depth image but at least 32 kB	None
Minimum supported PNG colour depth	32 bit per pixel	None
Minimum supported timer resolution	None	40 ms
Minimum supported number of timers per MIDlet suite	None	5

These requirements and recommendations apply to implementations of MSA and MSA Subset.