Technical Manual

V3x Handset

J2ME™ Developer Guide

Version 1.01

MOTOROLA
intelligence everywhere™

Table of Contents

Table of Figures

Index of Tables

Table of Code Samples

1 Introduction

Purpose

This document describes the application program interfaces used to develop Motorola compliant Java™ 2 Platform, Micro Edition (J2ME™) applications for the V3x handset.

Audience

This document is intended for premium J2ME developers and specific carriers involved with the development of J2ME applications for the V3x handset.

Disclaimer

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

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

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

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

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting form use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

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

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

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

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

References

Reference	Link
GSM 03.38 standard	http://www.etsi.org
GSM 03.40 standard	http://www.etsi.org
IETF RFC 2119	http://www.ietf.org/rfc/rfc2119.txt?number=2119
JSR 75	http://jcp.org/en/jsr/detail?id=75
JSR 205	http://www.jcp.org
MIDP 2.0	http://java.sun.com/products/midp/
RFC 2437	http://ietf.org/rfc/rfc2437.txt
Sun MIDP 2.0 SDK	http://java.sun.com/products/midp/
SSL protocol version 3.0	http://home.netscape.com/eng/ssl3/draft302.txt
Sun J2ME	http://java.sun.com/j2me/).
TLS protocol version 1.0	http://www.ietf.org/rfc/rfc2246.txt

Table 1 References

Revision History

Version	Date	Reason
01.00	Dez 26, 2005	Initial Release
01.01	Jan 24, 2006	Corrections in Spec Sheet and inclusion of JSR 139 chapter

Table 2 Revision History

Definitions, Abbreviations, Acronyms

Acronym	Description
3GPP	3rd Generation Partnership Project
AAC	Advanced Audio Coding
AMS	Application Management Software
API	Application Program Interface.
AUF	Application Utility Function
Bonding	Bonding is the creation of a relationship between two devices. The bond is a link key The relationship is created when the link key is exchanged between two devices. The devices are known to each other prior to the bonding procedure. A user initiates the bonding procedure and enters a passkey with the explicit purpose of creating a bond between two devices. This differs from the authenticate using a passkey procedure where the user is requested to enter a passkey during the establishment of the link.
BT	Bluetooth
CLDC	Connected Limited Device Configuration
DRM	Digital Rights Management
GPS	Global Positioning System
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
IDE	Integrated Development Environment
iTM	iTunes Mobile
ITU	International Telecommunication Union
JAD	Java Application Descriptor
JAL	Java Application Loader
JAR	Java Archive. Used by J2ME applications for compression and packaging.

J2ME	Java 2 Micro Edition
JSR	Java Specification Request
JSR 205	Java Specification Request 205. This JSR will extend and enhance the "Wireless Messaging API" (JSR-120)
KVM	K Virtual Machine (J2ME runtime environment)
L2CAP	Logical Link Control and Adaptation Protocol
Link Key	The authentication key used to establish a link between devices.
MP3	MPEG Layer 3
M4A	MPEG 4 Audio
M4P	MPEG 4 Audio - Protected (Fair-Play)
P2K	Platform 2000
Paired Device	A device with which a link key has been exchanged (either before connection establishment was requested or during connecting phase). See also pre-paired device and un-paired device.
SAA	SMS Access API
Trusted Device	A paired device that is explicitly marked as trusted.
UI	User Interface
VBR	Variable Bit Rate
VM	Virtual Machine
WAV	Wave Form Audio

Table 3 Definitions, Abbreviations, Acronyms

Document Overview

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

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

Chapter 2 – **J2ME Introduction:** this chapter describes the J2ME platform and the available resources on the V3x.

Chapter 3 – Developing and Packaging J2ME Applications: this chapter describes important features to look for when selecting tools and emulation environments. It also describes how to package a J2ME application, how to package a MIDlet, and generate JAR and JAD files properly.

Chapter 4 –Downloading Applications: this chapter describes the process for downloading applications.

Chapter 5 – Application Management: this chapter describes the lifecycle, installation/de-installation, and updating process for a MIDlet suite.

Chapter 6 – JSR 205 – WMA 2.0: this chapter describes the functionality that shall be implemented for the WMA.

Chapter 7 – JSR 120 – Wireless Messaging API: this chapter describes JSR 120 implementation.

Chapter 8 – JSR 135 – Mobile Media API: this chapter describes image types and supported formats.

Chapter 9 – JSR 75 – PIM and File Connection APIs: this chapter describes the JSR-75 APIs implementation requirements that shall replace the earlier implemented Phonebook and File Connection APIs requirements.

Chapter 10 – JSR 184 – 3D API: this chapter describes JSR 184.

Chapter 11 – JSR 185 – JTWI – this chapter describes JTWI functionality.

Chapter 12 – JSR 82 – Bluetooth API: this chapter describes the JSR-82, that covers the establishment of connections between devices for such applications as peer-to-peer gaming and Bluetooth pen use.

Chapter 13 – MIDP 2.0 Security Model: this chapter describes the MIDP 2.0 default security model.

Chapter 14 – Prevent Downloading of Large Java MIDlets : This feature makes flexible way of preventing the large JAR files OTA download.

Chapter 15 – Launch native streaming video client from Java application: This chapter how the feature delivers a capability for a Java application (MIDlet) to launch a native video streaming client on the handset.

Chapter 16 – JSR 139 – CLDC 1.1: this chapter describes briefly some characteristics of CLDC 1.1 and presents additional classes, fields, and methods supported for CLDC 1.1.

Appendix A – Key Mapping: this appendix describes the key mapping of the Motorola V3x, including the key name, key code, and game action of all Motorola keys.

Appendix B – Memory Management Calculation: this appendix describes the memory management calculations.

Appendix C – FAQ: this appendix provides a link to the dynamic online FAQ.

Appendix F – Spec Sheet: this appendix provides the spec sheet for the Motorola V3x handset.

Appendix H – Quick Reference: this appendix provides quick references to this document.

2
J2ME Introduction

The V3x includes the Java™ 2 Platform, Micro Edition, also known as the J2ME platform. The J2ME platform enables developers to easily create a variety of Java applications ranging from business applications to games. Prior to its inclusion, services or applications residing on small consumer devices like cell phones could not be upgraded or added to without significant effort. By implementing the J2ME platform on devices like the V3x, service providers, as well as customers, can easily add and remove applications allowing for quick and easy personalization of each device. This chapter of the guide presents a quick overview of the J2ME environment and the tools that can be used to develop applications for the V3x.

The Java 2 Platform, Micro Edition (J2ME)

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

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

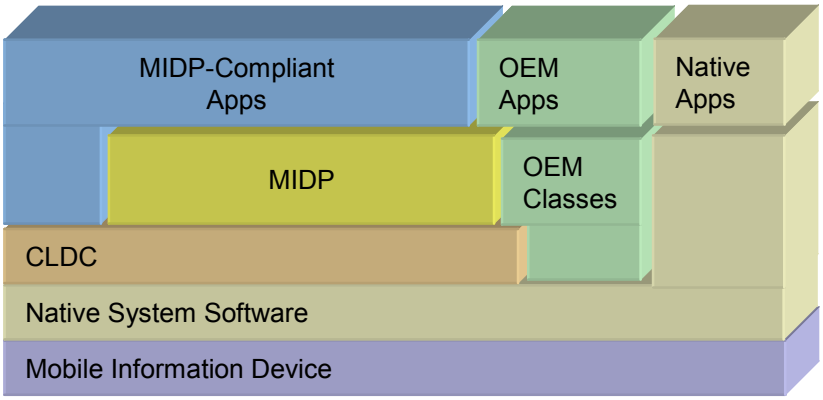

Figure 1 Java Platform

The Configuration Layer used in the V3x is the Connected Limited Device Configuration 1.1 (CLDC 1.1) and the Profile Layer used is the Mobile Information Device Profile 2.0 (MIDP 2.0). Together, the CLDC and MIDP provide common APIs for I/O, simple math functionality, UI, and more.

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

The Motorola J2ME Platform

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

The V3x contains OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the V3x can run any application written in standard MIDP, it can also run applications that take advantage of the unique functionality provided by these APIs. These OEM APIs are described in this guide.

Resources and API's Available

MIDP 2.0 will provide support to the following functional areas on the V3x:

MIDP 2.0

- Application delivery and billing
- Application lifecycle
- Application signing model and privileged security model
- End-to-end transactional security (HTTPS)
- MIDlet push registration (server push model)
- Networking

- Persistent storage
- Sounds
- Timers
- User Interface
- File Image Support (.PNG, .JPEG, .GIF)

Additional Functionality

- WMA (JSR 120)
- MMA (JSR 135)
- JSR 205
- JSR 184
- JSR 185
- JSR 82
- JSR 118

3
Developing and Packaging J2ME Applications

Guide to Development in J2ME

Introduction to Development

This appendix assumes the reader has previous experience in J2ME development and can appreciate the development process for Java MIDlets. This appendix will provide some information that a beginner in development can use to gain an understanding of MIDlets for J2ME handsets.

There is a wealth of material on this subject on websites maintained by Motorola, Sun Microsystems and others. Please refer to the following URLs for more information:

- http://www.motocoder.com
- http://www.java.sun.com/j2me
- http://www.corej2me.com/
- http://www.javaworld.com/

As an introduction, brief details of J2ME are explained below.

The MIDlet will consist of two core specifications, namely Connected, Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). Both of these specifications (Java Specification Requests) can be located at the http://www.jcp.org/ site for reading.

- For MIDP 1.0; JSR 37 should be reviewed.
- For MIDP 2.0; JSR 118 should be reviewed.
- For CLDC 1.0.4; JSR 30 should be reviewed.

To determine what implementation is on Motorola handset, review the “Java System” details through the menu on the Motorola handset (located under Java Settings).

For beginning development, key points to remember are memory size, processing power, screen capabilities and wireless network characteristics. These all play an important part in development of a MIDlet. The specifications listed above are designed to work upon devices that have these characteristics.

Network conditions would only apply for networked applications such as streaming tickers, email clients, etc.

In addition to the specifications, an array of tools is available to assist the development cycle. These range from the command line tools provided with Software Development Kits (SDK) from Sun (as of writing 1.4.1_04) to Integrated Development Environments (IDEs) which can be free or purchased. These IDEs come from a range of sources such as Sun, IBM, Metrowerks and Borland to name a few.

For a look at such environments, review the “Motorola T720 Handset Developer Guide” which is available from the MOTOCODER website.

In addition to the IDEs and Sun SDK for development, Motorola offers access to our own SDK which contains Motorola device emulators. From here, a MIDlet can be built and then deployed onto an emulated target handset. This will enable debugging and validation of the MIDlet before deployment to a real, physical handset. The latest Motorola SDK can be downloaded from the MOTOCODER website.

Please refer to the product specifications at the back of this guide for detailed information on each handset.

4
Downloading Applications

Methods of Downloading

There are two options open to the developer for deploying the MIDlet to a physical Motorola device. These are OTA (over -the-air) downloading or direct cable (Serial) downloading through a PC to the target device.

Method 1 - OTA

To use the OTA method, the developer will have a connection through a wireless network to a content server. This content server could be, for example, Apache (http://httpd.apache.org) which is free to use, deployable on multiple operating systems, and has extensive documentation on how to configure the platform.

The required file will be downloaded (either .jad and/or .jar) by issuing a direct URL request to the file in question or it could be a URL request to a WAP page and a hyperlink on that page to the target file. This request will be made through the Motorola Internet Browser (MIB). In MIDP 2.0, the need for a JAD file before download is not required, so the JAR file can be downloaded directly. The information about the MIDlet will be pulled from the manifest file.

The transport mechanism used to download the file will be one of two depending on the support from the network operators WAP Gateway and the size of file requested.

- HTTP Range – see specification RFC 2068 at http://www.rfc-editor.org/rfc.html if content greater than 30k in size. Below is a ladder diagram showing the flow through HTTP range transfer, although recall use of the .JAD is optional.
- SAR (Segmentation & Reassembly) – see specification of wireless transaction protocol at the http://www.wapforum.org if less than 30k in size.

During a download of the application, the user will see the MIB 2.2.1 browser displaying 'Downloading' followed by "x% completed" for either SAR or HTTP Byte Range type downloads.

A complete guide for setting up an OTA server can be obtained through the MOTOCODER website (http://www.motocoder.com). This includes details of configuring the server and also example WAP pages.

In these handsets, the user is given an option of deleting any MIDlets that are on the phone if an OTA download cannot be achieved due to lack of space.

The following error codes are supported:

- 900 Success
- 901 Insufficient Memory
- 902 User Cancelled
- 903 Loss Of Service
- 904 JAR Size Mismatch
- 905 Attribute Mismatch
- 906 Invalid Descriptor
- 907 Invalid JAR
- 908 Incompatible Configuration or Profile
- 909 Application Authentication Failure
- 910 Application Authorization Failure
- 911 Push Registration Failure
- 912 Deletion Notification

Please be aware that the method used by the handset, as per the specifications, is POST. Using a GET (URL encoding) style for the URL will fail. This is not the correct use of the MIDlets JAD parameters.

Possible Screen Messages Seen With Downloading:

- If JAR -file size does not match with specified size, it displays “Failed Invalid File”. Upon time-out, the handset goes back to browser.

- When downloading is done, the handset displays a transient notice “Download Completed” and starts to install the application.

- Upon completing installation, the handset displays a transient notice “Installed” and returns to Browser after time-out.

- If the MANIFEST file is wrong, the handset displays a transient notice “Failed File Corrupt” and returns to Browser after time-out.

If JAD does not contain mandatory attributes, “Failed Invalid File” notice appears

Method 2 - Direct Cable & Motorola MIDway Tool

The direct cable approach can be performed using a tool available from MOTOCODER called MIDway. The version available of writing is 2.8, which supports USB cable download. In order to use the tool properly, review FAQ 64 which contains the following about downloading:

1. MIDway 2.8.x executable
2. USB Driver for the cable to handset
3. Instructions on installation
4. User Guide for 2.8.x MIDway

In addition to the software the following parts will also be needed:

- USB Cable Part Number (SKN6371B).

It is important to note that the MIDway tool will only work with a device that has been enabled to support direct cable Java download. This feature is not available by purchasing a device through a standard consumer outlet.

The easiest method of confirming support for this is by looking at the “Java Tool” menu (see Figure 2) on the phone in question and seeing if a “Java app loader” option is available on that menu. If it is not, then contact MOTOCODER support for advice on how to receive an enabled handset.

Motorola provides a User Guide with the MIDway tool (as listed above) as well as a document outlining the tool for version 2.8 on the MOTOCODER website entitled “Installing J2ME MIDlet using MIDway Tool”.

Figure 2 MIDway "Java Tool" menu.

The USER-AGENT String

The Table 4 describes USER_AGENT strings associated with Motorola devices:

Motorola Device	USER_AGENT STRING
V3x	`User-Agent: MOT-V3x/xx.xx.xxR MIB/2.2.1 Profile/MIDP-2.0 Configuration/CLDC-1.1`

Table 4 USER_AGENT String

The USER_AGENT string can be used to identify a handset and render specific content to it based on it information provided in this string (example CGI on a content server). These strings can be found in the connection logs at the content server.

This table above provides information about the following components:

1. WAP Browser Release, Motorola Internet Browser (MIB) 2.2.1
2. MIDP version 2.0
3. CLDC version 1.1

Error Logs

The Table 5 represents the error logs associated with downloading applications.

Error Dialog	Scenario	Possible Cause	Install-Notify
Failed: Invalid File	JAD Download	Missing or incorrectly formatted mandatory JAD attributes Mandatory: MIDlet-Name (up to 32 symbols) MIDlet-Version MIDlet-Vendor (up to 32 symbols) MIDlet-JAR-URL (up to 256 symbols) MIDlet-JAR_Size	906 Invalid descriptor
Download Failed	OTA JAR Download	The received JAR size does not match the size indicated	904 JAR Size Mismatch
Cancelled: <Icon> <Filename>	OTA JAR Download	User cancelled download	902 User Cancelled
Download Failed	OTA JAR Download	Browser lost connection with server Certification path cannot be validated JAD signature verification failed Unknown error during JAD validation See 'Details' field in the dialog for information about specific error	903 Loss of Service
Insufficient Storage	OTA JAR Download	Insufficient data space to temporarily store the JAR file	901 Insufficient Memory
Application Already Exists	OTA JAR Download	MIDlet version numbers are identical	905 Attribute Mismatch
Different Version Exists	OTA JAR Download	MIDlet version on handset supercedes version being downloaded	
Failed File Corrupt	Installation	Attributes are not identical to respective JAD attributes	
Insufficient Storage	Installation	Insufficient Program Space or Data Space to install suite	901 Insufficient Memory
Application Error	Installation	Class references non-existent class or method Security Certificate verification failure Checksum of JAR file is not equal to Checksum in MIDlet-JAR-SHA attribute	

		Application not authorized
Application Expired	MIDlet Launching	Security Certificates expired or removed
Application Error	MIDlet Execution	Authorization failure during MIDlet execution Incorrect MIDlet

Table 5 Error Logs

5
Application Management

The following sections describe the application management scheme for the Motorola V3x handset. This chapter will discuss the following:

- Downloading a JAR without a JAD
- MIDlet upgrade
- Installation and Deletion Status Reports

Downloading a JAR file without a JAD

In Motorola's MIDP 2.0 implementation, a JAR file can be downloaded without a JAD. In this case, the user clicks on a link for a JAR file, the file is downloaded, and a confirmation will be obtained before the installation begins. The information presented is obtained from the JAR manifest instead of the JAD.

MIDlet Upgrade

Rules from the JSR 118 will be followed to help determine if the data from an old MIDlet should be preserved during a MIDlet upgrade. When these rules cannot determine if the RMS should be preserved, the user will be given an option to preserve the data.

The following conditions are used to determine if data can be saved:

- The data is saved if the new MIDlet-version is the same or newer, and if the new MIDlet-data-space requirements is the same or more than the current MIDlet.
- The data is not saved if the new MIDlet-data-space requirement is smaller than the current MIDlet requirement.
- The data is not saved if the new MIDlet-version is older than the current version.

If the data cannot be saved, the user will be warned about losing data. If the user proceeds, the application will be downloaded. If the user decides to save the data from the current MIDlet, the data will be preserved during the upgrade and the data will be made available for the new application. In any case, an unsigned MIDlet will not be allowed to update a signed MIDlet.

Installation and Deletion Status Reports

The status (success or failure) of an installation, upgrade, or deletion of a MIDlet suite will be sent to the server according to the JSR 118 specification. If the status report cannot be sent, the MIDlet suite will still be enabled and the user will be allowed to use it. In some instances, if the status report cannot be sent, the MIDlet will be deleted by operator's request. Upon successful deletion, the handset will send the status code 912 to the MIDlet-Delete-Notify URL. If this notification fails, the MIDlet suite will still be deleted. If this notification cannot be sent due to lack of network connectivity, the notification will be sent at the next available network connection.

Refer to the Table 6 for application management feature/class support for MIDP 2.0:

Feature/Class
Application upgrades performed directly through the AMS
When removing a MIDlet suite, the user will be prompted to confirm the entire MIDlet suite will be removed
Application Descriptor included the attribute MIDlet-Delete-Confirm, its value will be included in the prompt
Prompt for user approval when the user has chosen to download an application that is identical to, or an older version of an application currently in the handset
Unauthorized MIDlets will not have access to any restricted function call
AMS will check the JAD for security indicated every time a download is initiated
Application descriptor or MIDlet fails the security check, the AMS will prevent the installation of that application and notify the user that the MIDlet could not be installed
Application descriptor and MIDlet pass the security check , the AMS will install the MIDlet and grant it the permissions specified in the JAD
A method for launching Java application that maintains the same look and feel as other features on the device will be provided
User will be informed of download and installation with a single progress indicator and will be given an opportunity to cancel the process
User will be prompted to launch the MIDlet after installation
A method for creating shortcuts to Java applications will be provided.
A no forward policy on DRM issues, included but not limited to transferring the application over-the-air, IRDA, Bluetooth, I/O Cables, External storage devices, etc until further guidance is provided

Table 6 application management feature

6
JSR 205 – WMA 2.0

Overview

This section describes the functionality that shall be implemented for the WMA. This section highlights implementation details with respect to the messaging API that are important to this implementation rather than restating entire JSR-205; refer the JSR 205 for more details. This section also provides Motorola specific requirements for WMA in addition to JSR-205.

Any Motorola device implementing JTWI will support the following minimum hardware requirements in addition to the minimum requirements specified in MIDP 2.0:

- At least a 125 x 125 pixels screen size as returned by full screen mode `Canvas.getHeight ()` and `Canvas.getWidth ()`
- At least a color depth of 4096 colors (12-bit) as returned by Display.numColors ()
- Pixel shape of 1:1 ratio
- At least a Java Heap Size of 512 KB
- Sound mixer with at least 2 sounds
- At least a JAD file size of 5 KB
- At least a JAR file size of 64 KB
- At least a RMS data size of 30 KB

Any Motorola JTWI device will implement the following and pass the corresponding TCK:

- CLDC 1.1
- MIDP 2.0 (JSR 118)
- Wireless Messaging API 1.1 (JSR 120)
- Mobile Media API 1.1 (JSR 135)

CLDC related content for JTWI

JTWI is designed to be implemented on top of CLDC 1.0 or CLDC 1.1. The configuration provides the VM and the basic APIs of the application environment. If floating point capabilities are exposed to Java Applications, CLDC 1.1 (not suported on V3x) will be implemented.

The following CLDC requirements will be supported:

- Minimum Application thread count will allow a MIDlet suite to create a minimum of 10 simultaneous running threads
- Minimum Clock Resolution – The method java.land.System.currentTimeMillis () will record the elapsed time in increments not to exceed 40 msec. At least 80% of test attemps will meet the time elapsed requirement to achieve acceptable conformance.
- Names for Encodings will support at least the preferred MIME name as defined by IANA (http://www.iana.org/assignments/character-sets) for the supported character encodings. If not preferred name has been defined, the registered name will be used (i.e UTF-16).
- Character Properties will provide support for character properties and case conversions for the characters in the Basic Latin and Latin-1 Supplement blocks of Unicode 3.0. Other Unicode character blocks will be supported as necessary.
- Unicode Version will support Unicode characters. Character information is based on the Unicode Standard version 3.0. Since the full character tables required for Unicode support can be excessively large for devices with tight memory budgets, by default, the character property and case conversion facilities in CLDC assume the presence of ISO Latin-1 range of characters only. Refer to JSR 185 for more information.
- Custom Time Zone Ids will permit use of custom time zones which adhere to the following time xone format:
    - General Time Zone: For time zones representing a GMT offset value, the following syntax is used:
        - Custom ID:
            - GMT Sign Hours: Minutes
            - GMT Sign Hours Minutes
            - GMT Sign Hours Hours
        - Sign: one of:
            - + -
        - Hours:
            - Digit
            - Digit Digit
        - Minutes:

        - Digit Digit
    - Digit: one of:
        - 0 1 2 3 4 5 6 7 8 9

NOTE: Hours will be between 0 and 23, and minutes will be between 00 and 50. For example, GMT +10 and GMT +0010 equates to ten hours and ten minutes ahead of GMT.

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

MIDP 2.0 specific information for JTWI

MIDP 2.0 provides the library support for user interface, persistent storage, networking, security, and push functions. MIDP 2.0 contains a number of optional functions, some of which will be implemented as outlined below. The following JTWI requirements for MIDP 2.0 will be supported:

- Record Store Minimum will permit a MIDlet suite to create at least 5 independent RecordStores. This requirement does not intend to mandate that memory be reserved for these Record Stores, but it will be possible to create the RecordStores if the required memory is available.
- HTTP Support for Media Content will provide support for HTTP 1.1 for all supported media types. HTTP 1.1 conformance will match the MIDP 2.0 specification. See package.javax.microedition.io for specific requirements.
- JPEG for Image Objects – ISO/IEC JPEG together wil JFIF will be supported. The support for ISO/IEC JPEG only applies to baseline DCT, non-differential, Huffman coding, as defined in JSR 185 JTWI specification, symbol ‘SOF0’. This support extends to the class javax.microedition.lcdui.Image, including the methods outlined above. This mandate is voided in the event that the JPEG image format becomes encumbered with licensing requirements.

- Timer Resolution will 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. At least 80% of test attempts will meet the schedule resolution requirement to achieve acceptable conformance.
- Minimum Number of Timers will allow a MIDlet to create a minimum of 5 simultaneously running Timers. This requirement is independent of the minimum specified by the Minimum Application Thread Count.
- Bitmap Minimums will 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 format specification. For each of these color depths, as well as for JFIF image formats, a compliant implementation will support images up to 76800 total pixels.
- TextField and TextBox and Phonebook Coupling – when the center select key is pressed while in a TextBox or TextField and the constraint of the TextBox or TextField is TextField.PHONENUMBER, the names in the Phonebook will be displayed in the “Insert Phonenumber?” screen.
- Supported characters in TextField and TextBox – TextBox and TextField with input constraint TextField.ANY will support inputting all the characters listed in JSR 185.
- Supported characters in EMAILADDR and URL Fields – Class javax.microedition.lcdui.TextBox and javax.microedition.lcdui.TextField with either of the constraints TextField.EMAILADDR or TextField.URL will allow the same characters to be input as are allowed for input constraint TextField.ANY
- Push Registry Alarm Events will implement alarm-based push registry entries.
- Identification of JTWI via system property – to identify a compliant device and the implemented version of this specification, the value of the system property microedition.jtwi.version will be 1.0

Wireless Messaging API 1.1 (JSR 120) specific content for JTWI

WMA defines an API used to send and receive short messages. The API provides access to network-specific short message services such as GSM SMS or CDMA short messaging. JTWI will support the following as it is outlined in the JSR 120 chapter of this developer guide:

- Support for SMS in GSM devices
- SMS Push

Mobile Media API 1.1 (JSR 135) specific content for JTWI

The following will be supported for JTWI compliance:

- HTTP 1.1 Protocol will be supported for media file download for all supported media formats
- MIDI feature set specified in MMAPI (JSR 135) will be implemented. MIDI file playback will be supported.
- VolumeControl will be implemented and is required for controlling the volume of MIDI file playback.
- JPEG encoding in video snapshots will be supported if the handset supports the video feature set and video image capture.
- Tone sequence file format will be supported. Tone sequences provide an additional simple format for supporting the audio needs of many types of games and other applications.

MIDP 2.0 Security specific content for JTWI

The Motorola V3x follows the security policy outlined in the Security chapter of this developer guide.

Wireless Messaging API 2.0 (JSR 205)

This section describes the functionality that shall be implemented for the WMA. This section highlights implementation details with respect to the messaging API that are important to this implementation rather than restating entire JSR-205; refer the JSR 205 for more details. This section also provides Motorola specific requirements for WMA in addition to JSR-205.

Messaging Functionality

This section describes messaging functionality to be implemented by WMA.

MMS Message Structure

The MMS PDU structure SHALL be implemented as specified in the WAP-209-MMSEncapsulation standard. The MMS PDU consists of headers and a multipart message body. Some of the headers originate from standard RFC 822 headers and

others are specific to multimedia messaging. In addition to defined MMS headers, it also contains header parameters as defined by JSR 205. The message body may contain parts of any content type and MIME multipart is used to represent and encode a wide variety of media types for transmission via multimedia messaging.

MMS Message Addressing

The multipart message addressing model contains different types of addresses:

- global telephone number of recipient user, including telephone number, ipv4, ipv6 adresses
- e-mail address as specified in RFC822
- short-code of the service (not valid for MMS version 1.0)

The syntax of the URL connection strings SHALL follow the rules specified in Table D-2 of JSR- 205 specification.

MMS Message Types

MMS messages can be sent using MULTIPART_MESSAGE type of this API. The default type of message is multipart/related. If the content type header does not contain start parameter, the message type is asumed to be multipart/mixed. This section describes Multipart Message, and its related classes. Messaging framework is described in the JSR 120 chapter of this developer guide.

MultipartMessage

The WMA shall implement the MultipartMessage an interface representing a multipart message. This is a sub interface of Message which contains methods to add, remove and manipulate message parts. The interface also allows to specify the subject of the message.

Please refer JSR 205 specification for more details.

MessagePart

The WMA shall implement the MessagePart a class representing a media part that can be

sent with the message. Instances of MessagePart class are added to the MultipartMessage.

Each message part consists of part header and part body. The part headers include Content ID, Content Location, Content type, Encoding scheme. Content can be of any MIME type.

Multimedia Message Service Center Address

The MMSC address used for sending the messages should be made available using System.getProperty with property name “wireless.messaging.mms.mmsc”. Applications might need to obtain the Multimedia Message Service Center (MMSC) address to decide which recipient to use. For example, the application might need to do this because it is using service numbers for application servers which might not be consistent in all networks and MMSCs.

Please refer section D.3.0 of JSR 205 specification for more details.

Application ID

The WMA supports sending of MMS messages to concrete Java application. To enable this the following additional parameters SHALL be added to Content-Type header field:

Messages can be sent using this API via client or server type Message Connections, refer section D.2.2 of JSR-205 specification.

The application specifies Application-ID when opening the server mode MessageConnection. The receiving application running on a device is identified with the application-id included in the message.

The maximum number of Application-IDs shall be limited by the implementation and depends on phone RAM availability and carrier operators preloaded content memory consumption.

The maximum number of simultaneously opened connections shall be limited by the implementation and depends on phone RAM availability and carrier operators preloaded content memory consumption.

The maximum number of MMS messages in the buffer at the same time shall be limited by the implementation and depends on phone RAM availability and carrier operators preloaded content memory consumption.

7
JSR 120 – Wireless Messaging API

Wireless Messaging API (WMA)

Motorola has implemented certain features that are defined in the Wireless Messaging API (WMA) 1.0 and 1.3 versions. The complete specification document is defined in JSR 120.

The JSR 120 specification states that developers can be provided access to send (MO – mobile originated) and receive (MT – mobile terminated) SMS (Short Message Service) on the target device.

A simple example of the WMA is the ability of two J2ME applications using SMS to communicate game moves running on the handsets. This can take the form of chess moves being passed between two players via the WMA.

Motorola in this implementation of the specification supports the following features.

- Creating an SMS
- Sending an SMS
- Receiving an SMS
- Viewing an SMS
- Deleting an SMS

SMS Client Mode and Server Mode Connection

The Wireless Messaging API is based on the Generic Connection Framework (GCF), which is defined in the CLDC specification 1.0. The use of the “Connection” framework, in Motorola's case is “`MessageConnection`”.

The `MessageConnection` can be opened in either server or client mode. A server connection is opened by providing a URL that specifies an identifier (port number) for an application on the local device for incoming messages.

```
(MessageConnection)Connector.open("sms://:6000");
```

Messages received with this identifier will then be delivered to the application by this connection. A server mode connection can be used for both sending and receiving messages. A client mode connection is opened by providing a URL which points to another device. A client mode connection can only be used for sending messages.

```
(MessageConnection)Connector.open("sms://+441234567890
:6000");
```

SMS Port Numbers

When a port number is present in the address, the TP-User-Data of the SMS will contain a User-Data-Header with the application port addressing scheme information element. When the recipient address does not contain a port number, the TP-User-Data will not contain the application port addressing header. The J2ME MIDlet cannot receive this kind of message, but the SMS will be handled in the usual manner for a standard SMS to the device.

When a message identifying a port number is sent from a server type `MessageConnection`, the originating port number in the message is set to the port number of the `MessageConnection`. This allows the recipient to send a response to the message that will be received by this `MessageConnection`.

However, when a client type `MessageConnection` is used for sending a message with a port number, the originating port number is set to an implementation specific value and any possible messages received to this port number are not delivered to the `MessageConnection`. Please refer to the section A.4.0 and A.6.0 of the JSR 120.

When a MIDlet in server mode requests a port number (identifier) to use and it is the first MIDlet to request this identifier it will be allocated. If other applications apply for the same identifier then an `IOException` will be thrown when an attempt to open `MessageConnection` is made. If a system application is using this identifier, the MIDlet will not be allocated the identifier. The port numbers allowed for this request are restricted to SMS messages. In addition, a MIDlet is not allowed to send messages to certain restricted ports a `SecurityException` will be thrown if this is attempted.

JSR 120 Section A.6.0 Restricted Ports:
2805, 2923, 2948, 2949, 5502, 5503, 5508, 5511, 5512, 9200, 9201, 9203, 9207, 49996, 49999.

If you intend to use SMSC numbers then please review A.3.0 in the JSR 120 specification. The use of an SMSC would be used if the MIDlet had to determine what recipient number to use.

SMS Storing and Deleting Received Messages

When SMS messages are received by the MIDlet, they are removed from the SIM card memory where they were stored. The storage location (inbox) for the SMS messages has a capacity of up to thirty messages. If any messages are older than five days then this will be removed, from the inbox by way of a FIFO stack.

SMS Message Types

The types of messages that can be sent are TEXT or BINARY, the method of encoding the messages are defined in GSM 03.38 standard (Part 4 SMS Data Coding Scheme). Refer to section A.5.0 of JSR 120 for more information.

SMS Message Structure

The message structure of SMS will comply with GSM 03.40 v7.4.0 Digital cellular telecommunications system (Phase 2+); Technical realization of the Short Message Service (SMS) ETSI 2000.

Motorola's implementation uses the concatenation feature specified in sections 9.2.3.24.1 and 9.2.3.24.8 of the GSM 03.40 standard for messages that the Java application sends that are too long to fit in a single SMS protocol message.

This implementation automatically concatenates the received SMS protocol messages and passes the fully reassembled message to the application via the API. The implementation will support at least three SMS messages to be received and concatenated together. Also, for sending, support for a minimum of three messages is supported. Motorola advises that developers will not send messages that will take up more than three SMS protocol messages unless the recipient's device is known to support more.

SMS Notification

Examples of SMS interaction with a MIDlet would be the following:

- A MIDlet will handle an incoming SMS message if the MIDlet is registered to receive messages on the port (identifier) and is running.

- When a MIDlet is paused and is registered to receive messages on the port number of the incoming message, then the user will be queried to launch the MIDlet.
- If the MIDlet is not running and the Java Virtual Machine is not initialized, then a Push Registry will be used to initialize the Virtual Machine and launch the J2ME MIDlet. This only applies to trusted, signed MIDlets.
- If a message is received and the untrusted unsigned application and the KVM are not running then the message will be discarded.
- There is a SMS Access setting in the Java Settings menu option on the handset that allows the user to specify when and how often to ask for authorization. Before the connection is made from the MIDlet, the options available are:
    - Always ask for user authorization
    - Ask once per application
    - Never Ask

The table 7 is a list of Messaging features/classes supported in the device.

Feature/Class	Implementation
JSR-120 API. Specifically, APIs defined in the javax.wireless.messaging package will be implemented with regards to the GSM SMS Adaptor	Supported
Removal of SMS messages	Supported
Terminated SMS removal – any user prompts handled by MIDlet	Supported
Originated SMS removal – any user prompts handled by MIDlet	Supported
All fields, methods, and inherited methods for the Connector Class in the javax.microedition.io package	Supported
All methods for the BinaryMessage interface in the javax.wireless.messaging package	Supported
All methods for the Message interface in the javax.wireless.messaging package	Supported
All fields, methods, and inherited methods for the MessageConnection interface in the javax.wireless.messaging package	Supported
Number of MessageConnection instances in the javax.wireless.messaging package	32 maximum
Number of MessageConnection instances in the javax.wireless.messaging package	16
All methods for the MessageListener interface in the javax.wireless.messaging package	Supported
All methods and inherited methods for the TextMessage interface in the javax.wireless.messaging package	Supported

16 bit reference number in concatenated messages	Supported
Number of concatenated messages.	30 messages in inbox, each can be concatenated from 3 parts. No limitation on outbox (immediately transmitted)
Allow MIDlets to obtain the SMSC address with the wireless.messaging.sms.smsc system property	Supported

Table 7 List of Messaging features/classes supported

The code sample 1 shows implementation of the JSR 120 Wireless Messaging API:

Creation of client connection and for calling of method 'numberOfSegments' for Binary message:

```
BinaryMessage binMsg;
MessageConnection connClient;
int MsgLength = 140;

   /* Create connection for client mode */
   connClient = (MessageConnection) Connector.open("sms://" +
outAddr);

   /* Create BinaryMessage for client mode */
   binMsg =
(BinaryMessage)connClient.newMessage(MessageConnection.BINARY
_MESSAGE);

   /* Create BINARY of 'size' bytes for BinaryMsg */
   public byte[] createBinary(int size) {
      int nextByte = 0;
byte[] newBin = new byte[size];

      for (int i = 0; i < size; i++) {
         nextByte = (rand.nextInt());
         newBin[i] = (byte)nextByte;
         if ((size > 4) && (i == size / 2)) {
            newBin[i-1] = 0x1b;
            newBin[i] = 0x7f;
         }
      }
      return newBin;
   }

byte[] newBin = createBinary(msgLength);
   binMsg.setPayloadData(newBin);

int num = connClient.numberOfSegments(binMsg);
```

Creation of server connection:

```
MessageConnection messageConnection =
(MessageConnection)Connector.open("sms://:9532");
```

Creation of client connection with port number:

```
MessageConnection messageConnection =
(MessageConnection)Connector.open("sms://+18473297274:9532");
```

Creation of client connection without port number:

```
MessageConnection messageConnection =
(MessageConnection)Connector.open("sms://+18473297274");
```

Closing of connection:

```
MessageConnection messageConnection.close();
```

Creation of SMS message:

```
Message textMessage =
messageConnection.newMessage(MessageConnection.TEXT_MESSAGE);
```

Setting of payload text for text message:

```
((TextMessage)message).setPayloadText("Text Message");
```

Getting of payload text of received text message:

```
receivedText =
((TextMessage)receivedMessage).getPayloadText();
```

Getting of payload data of received binary message:

```
BinaryMessage binMsg;
byte[] payloadData = binMsg.getPayloadData();
```

Setting of address with port number:

```
message.setAddress("sms://+18473297274:9532");
```

Setting of address without port number:

```
message.setAddress("sms://+18473297274");
```

Sending of message:

```
messageConnection.send(message);
```

Receiving of message:

```
Message receivedMessage = messageConnection.receive();
```

Getting of address:

```
String address = ((TextMessage)message).getAddress();
```

Getting of SMS service center address via calling of System.getProperty():

```
String addrSMSC =
System.getProperty("wireless.messaging.sms.smsc");
```

Getting of timestamp for the message:

```
Message message;
System.out.println("Timestamp: " +
message.getTimestamp().getTime());
```

Creation of client connection, creation of binary message, setting of payload for binary message and calling of method 'numberOfSegments(Message)' for Binary message:

```
BinaryMessage binMsg;
MessageConnection connClient;
int MsgLength = 140;

   /* Create connection for client mode */
   connClient = (MessageConnection) Connector.open("sms://" +
outAddr);

   /* Create BinaryMessage for client mode */
   binMsg =
(BinaryMessage)connClient.newMessage(MessageConnection.BINARY
_MESSAGE);

   /* Create BINARY of 'size' bytes for BinaryMsg */
   public byte[] createBinary(int size) {
      int nextByte = 0;
byte[] newBin = new byte[size];

      for (int i = 0; i < size; i++) {
         nextByte = (rand.nextInt());
         newBin[i] = (byte)nextByte;
         if ((size > 4) && (i == size / 2)) {
            newBin[i-1] = 0x1b;
            newBin[i] = 0x7f;
         }
      }
      return newBin;
   }

byte[] newBin = createBinary(msgLength);
   binMsg.setPayloadData(newBin);
```

```
int num = connClient.numberOfSegments(binMsg);
```

Setting of MessageListener and receiving of notifications about incoming messages:

```
public class JSR120Sample1 extends MIDlet implements
CommandListener {
…
JSR120Sample1Listener listener = new JSR120Sample1Listener();
…
// open connection
messageConnection =
(MessageConnection)Connector.open("sms://:9532");
…
// create message to send
…
listener.run();
…
// set payload for the message to send
…
// set address for the message to send
messageToSend.setAddress("sms://+18473297274:9532");
…
// send message (via invocation of ‘send’ method)
…
// set address for the message to receive
receivedMessage.setAddress("sms://:9532");
…
// receive message (via invocation of ‘receive’ method)
…


class JSR120Sample1Listener implements MessageListener,
Runnable {
private int messages = 0;

public void notifyIncomingMessage(MessageConnection
connection) {
System.out.println("Notification about incoming message
arrived");
   messages++;
}

public void run() {
      try {
      messageConnection.setMessageListener(listener);
      } catch (IOException e) {
         result = FAIL;
System.out.println("FAILED: exception while setting listener:
" + e.toString());
   }
}
```

```
        }
```

Code Sample 1 JSR 120 Wireless Messaging API

App Inbox Clean-up

Actually, messages for MIDlets are stored in a separate App Inbox. This App Inbox is cleaned up automatically.

The App Inbox capacity is 26 messages or 26 segments and when a new message is received for a certain port number, and the App Inbox capacity has reached its limit of 26 messages, then the messages in the App Inbox will be deleted in the following order:

- If a certain port number has any unread messages in the App Inbox, then the oldest unread message in the buffer relative to that port number WILL be deleted next.
- If a certain port number currently has no messages in the App Inbox, then the oldest unread message in the buffer relative to all port numbers will be deleted next.

When the maximum number of messages is reached and the phone has reached memory full condition, no new messages can be received by the applications. A blinking messaging icon is used to inform the user that the messaging folder is full. At this stage the user has to manually delete some messages to clear some memory to allow the reception of incoming messages.

8
JSR 135 – Mobile Media API

JSR 135 Mobile Media API

The JSR 135 Mobile Media APIs feature sets are defined for five different types of media. The media defined is as follows:

- Tone Sequence
- Sampled Audio
- MIDI

The new implementation of JSR 135 supports playback of more audio formats and recording of time-based media – audio and video as well as still-image capture.

When a player is created for a particular type, it will follow the guidelines and control types listed in the sections outlined below.

The code sample 2 shows implementation of the JSR 135 Mobile Media API:

JSR 135

```
Player player;

// Create a media player, associate it with a stream
containing media data
try
{
    player =
Manager.createPlayer(getClass().getResourceAsStream("MP3.mp3
"), "audio/mp3");
}
catch (Exception e)
{
    System.out.println("FAILED: exception for createPlayer:
" + e.toString());
}

// Obtain the information required to acquire the media
resources
```

```
            try
            {
                player.realize();
            }
            catch (MediaException e)
            {
                System.out.println("FAILED: exception for realize: " +
            e.toString());
            }

            // Acquire exclusive resources, fill buffers with media data
            try
            {
                player.prefetch();
            }
            catch (MediaException e)
            {
                System.out.println("FAILED: exception for prefetch: " +
            e.toString());
            }

            // Start the media playback
            try
            {
                player.start();
            }
            catch (MediaException e)
            {
                System.out.println("FAILED: exception for start: " +
            e.toString());
            }

            // Pause the media playback
            try
            {
                player.stop();
            }
            catch (MediaException e)
            {
                System.out.println("FAILED: exception for stop: " +
            e.toString());
            }

            // Release the resources
            player.close();
```

Code Sample 2 JSR 135 Mobile Media API

ToneControl

ToneControl is the interface to enable playback of a user-defined monotonic tone sequence. The JSR 135 Mobile Media API will implement public interface ToneControl.

A tone sequence is specified as a list of non-tone duration pairs and user-defined sequence blocks and is packaged as an array of bytes. The `setSequence()` method is used to input the sequence to the ToneControl.

The following is the available method for ToneControl:

`-setSequence (byte[] sequence)`: Sets the tone sequence

VolumeControl

VolumeControl is an interface for manipulating the audio volume of a Player. The JSR 135 Mobile Media API will implement public interface VolumeControl.

The following describes the different volume settings found within VolumeControl:

- Volume Settings - allows the output volume to be specified using an integer value that varies between 0 and 100. Depending on the application, this will need to be mapped to the volume level on the phone (0-7).
- Specifying Volume in the Level Scale - specifies volume in a linear scale. It ranges from 0 – 100 where 0 represents silence and 100 represents the highest volume available.
- Mute – setting mute on or off does not change the volume level returned by the getLevel. If mute is on, no audio signal is produced by the Player. If mute is off, an audio signal is produced and the volume is restored.

The following is a list of available methods with regards to VoumeControl:

`-getLevel`: Get the current volume setting.

`-isMuted`: Get the mute state of the signal associated with this VolumeControl.

`-setLevel (int level)`: Set the volume using a linear point scale with values between 0 and 100.

`-setMute (Boolean mute)`: Mute or unmute the Player associated with this VolumeControl.

StopTimeControl

StopTimeControl allows a specific preset sleep timer for a player. The JSR 135 Mobile Media API will implement public interface StopTimeControl.

The following is a list of available methods with regards to StopTimeControl:

`-getStopTime`: Gets the last value successfully by setStopTime.

`-setStopTime (long stopTime)`: Sets the media time at which you want the Player to stop.

Manager Class

Manager Class is the access point for obtaining system dependant resources such as players for multimedia processing. A Player is an object used to control and render media that is specific to the content type of the data. Manager provides access to an implementation specific mechanism for constructing Players. For convenience, Manager also provides a simplified method to generate simple tones. Primarily, the Multimedia API will provide a way to check available/supported content types.

Audio Media

The table 8 shows multimedia file formats are supported:

File Type	CODEC
WAV	PCM
WAV	ADPCM
SP MIDI	General MIDI
MIDI Type 1	General MIDI
iMelody	IMelody
CTG	CTG
MP3	MPEG-1 layer III
AMR	AMR
BAS	General MIDI

Table 8 Multimedia File Formats

The table 9 is a list of audio MIME types supported:

Category	Description	MIME Type
Audio	iMelody	`audio/imelody x-imelody imy x-imy`
	MIDI	`audio/midi x-midi mid x-mid sp-midi mobile-xmf`

	WAV	audio/wav x-wav
	MP3	audio/mp3 x-mp3 mpeg3 x-mpeg3 mpeg x-mpeg
	AMR/MP4	audio/amr x-amr mp4 x-mp4 3gpp
	AAC	audio/m4a

Table 9 Audio MIME types

Refer to the table 10 for multimedia feature/class support for JSR 135:

Feature/Class	Implementation
Media package found	Supported
Media control package	Supported
Media Protocol package	Streaming not supported
Control interface in javax.microedition.media	Supported
All methods for the Controllable interface in javax.microedition.media.control	Supported
All fields, methods, and inherited methods for the Player interface in javax.microedition.media	Supported
All fields and methods for the PlayerListener interface in javax.microedition.media	Supported
PlayerListener OEM event types for the PlayerListener interface	Standard types only
All fields, methods, and inherited methods for the Manager Class in javax.microedition.media	Supported
TONE_DEVICE_LOCATOR support in the Manager class of javax.microedition.media	Supported
TONE_DEVICE_LOCATOR content type will be audio/x-tone-seq	Supported
TONE_DEVICE_LOCATOR media locator will be device://tone	Supported
All constructors and inherited methods in javax.microedition.medi.MediaException	Supported
All fields and methods in the StopTimeControl interface in javax.microedition.media.control	Supported
All fields and methods in the ToneControl interface in javax.microedition.media.control	Supported
All methods in the VolumeControl interface in javax.microedition.media.control	Supported
Max volume of a MIDlet will not exceed the maximum speaker setting on the handset	Supported
Multiple SourceStreams for a DataSource	2

Table 10 Multimedia feature/class support for JSR 135

Note: The multimedia engine only supports prefetching 1 sound at a time, but 2 exceptions exist where 2 sounds can be prefetched at once. These exceptions are listed below:

1. Motorola provides the ability to play MIDI and WAV files simultaneously, but the MIDI track will be started first. The WAV file should have the following format: PCM 8,000 Khz; 8 Bit; Mono

2. When midi, iMelody, mix, and basetracks are involved, two instances of midi, iMelody, mix, or basetrack sessions can be prefetched at a time, although one of these instances has to be stopped. This is a strict requirement as (for example) two midi sounds cannot be played simultaneously.

Mobile Media Feature Sets

The table 11 lists the new packages, classes, fields and methods implemented for JSR 135. Appropriate exception shall be generated if the called method is not supported by the implementation. If a method is accessed without proper security permissions, security exception shall be thrown.

Packages	Classes	Methods	Comments & Requirements
javax.microedition.media.control	TempoControl (Applicable to MIDI/iMelody audio formats. Implementation guidance - SHOULD.)	setTempo()	Sets the current playback tempo. WILL implement a tempo range of 10 to 300 beats per minute.
		getTempo()	Gets the current playback tempo.
	PitchControl (Applicable to MIDI /iMelody audio formats. Implementation guidance - SHOULD)	getMaxPitch()	Gets the maximum playback pitch raise supported by the player. SHOULD implement a maximum playback pitch raise of 12,000 milli-semitones.

		getMinPitch()	Gets the minimum playback pitch raise supported by the player. SHOULD implement a minimum playback pitch raise of 12,000 millisemitones.
		getPitch()	Gets the current playback pitch raise.
		setPitch()	Sets the relative pitch raise.
	FramePositioningControl (Implementation guidance - SHOULD)	mapFrameToTime()	Converts the given frame number to the corresponding media time.
		mapTimeToFrame()	Converts the given media time to the corresponding frame number.
		seek()	Seeks to a given video frame.
		skip()	Skips a given number of frames from the current position.
	MIDIControl (Implementation guidance - SHOULD)	All fields & methods	

	RecordControl	All fields & methods	RecordControl controls the recording of media from a Player. Supports all methods. Required for audio capture functionality. Video capture support is optional. RecordControl is a protected API as specified in the Security section.
	VideoControl (Implementation guidance - SHOULD)	All fields & methods. getSnapshot() method **WILL** be supported if the VideoControl is implemented by an instance of camera device. If VideoControl is implemented by video player for video file/ stream playback, it is not mandatory to support get Snap Shot() method.	VideoControl controls the display of video. A Player which supports the playback of video WILL provide a VideoControl via its getControl and getControls methods.
	MetaDataControl	Implement all fields and methods. Support title, copyright, data, author keys for CODECs supporting these keys.	
javax.microedition.media	Player	All fields and methods	SHOULD allow a Player to use a different TimeBase other than its own. This is required for synchronization between multiple Media Players.
	PlayerListener	All fields and methods	SHOULD let applications register PlayerListener for receiving Player events.

	Manager	All fields and methods	WILL support file locator for local playback. For streaming, RTP locator needs to be supported. For camera, new device locator, “camera” has to be supported.
	TimeBase	getTime()	Gets the current time of this TimeBase.
javax.microedition.media.protocol	ContentDescriptor	getContentType()	Obtains a string that represents the content type for this descriptor.

Table 11 New packages, classes, fields and methods implemented for JSR 135

There are others features that can be considered, such as:

- Support synchronous mixing of two or more sound channels. MIDI+WAV is supported, but MIDI+MP3 is highly desirable.
- The classes Manager, DataSource and RecordControl interface accepts media locators. In addition to normal playback locators specified by JSR – 135, the following special locators are supported:
    - **RTP Locators** are supported for streaming media on devices supporting real time streaming using RTSP. This support will be available for audio and video streaming through Manager (for playback media stream).
    - **HTTP Locators** are supported for playing back media over network connections. This support should be available through Manager implementation.
      e.g.: Manager.createPlayer(“http://webserver/tune.mid”)
    - **File locators** are supported for playback and capture of media. This is specific to Motorola J2ME implementations supporting file system API and not as per JSR-135. The support should be available through Manager and RecordControl implementations.
      e.g.: Manager.createPlayer(“file://motorola/audio/sample.mid”)
    - Capture Locator is supported for audio and video devices. A new device “camera” **will** be defined and supported for camera device. Manager.createPlayer() call shall return camera player as a special type of video player. Camera player should implement VideoControl and should support taking snapShots using VideoControl.getSnapshot() method.
      e.g.: Manager.createPlayer(“capture://camera”)

Supported Multimedia File Types

The tables 12, 13 and 14 lists multimedia file types (with corresponding CODECs) that are supported in products that are JSR-135 compliant. The common guideline being all codecs and file types supported by native side should be accessible through JSR-135 implementation. The implementation of JSR-135 (and these tables) is updated every time a new file type and/or CODEC is released.

Image Media

File Type	CODEC	Functionality
JPEG/JFIF	JPEG	Capture
JPEG/EXIF	JPEG	Capture

Table 12 Image Media

Audio Media

File Type	CODEC	Functionality
AAC	AAC	Playback
WMA	Proprietary (Microsoft)	Playback
AU	PCM Mu-law	Playback
AIFF	PCM	Playback
XMF	General MIDI	Playback
AMR	AMR NB	Playback and Capture

Table 13 Audio Media

Video Media

File Type	CODEC	Functionality
MP4	H.263 (profile 0) or MPEG 4 with or without AMR/AAC audio.	Playback and Capture
3GP	H.263 (profile 0) or MPEG 4 with or without AMR/AAC audio.	Playback and Capture
RTF Streaming	RTP/RTSP/RTCP Streaming Engine	Playback
ASF	Proprietary (Microsoft)	Playback
WMV	Proprietary (Microsoft)	Playback
Windows Streaming	Proprietary (Microsoft)	Playback

Table 14 Video Media

Canned Sounds

The implementation supports predefined sounds which can be used by applications like games and alerts. Each predefined sound has an associated locator, which are used by the application.

e.g.: Manager.createPlayer (CANNED_SOUND_ALARM)

Security

Mobile Media API follows MIDP 2.0 security model. Recording functionality APIs need to be protected. Trusted third party and untrusted applications will utilize user permissions. Specific permission settings are detailed below.

Policy

The table 15 shows security policy will be flexed in per operator requirements at ship time of the handset.

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
Multimedia Record	**Ask once Per App**, Always Ask, Never Ask, No Access	**Always Ask**, Ask once Per App, Never Ask, No Access	Full Access	Full Access

Table 15 Security Police

Permissions

The table 16 lists individual permissions within Multimedia Record function group.

Permission	Protocol	Function Group
javax.microedition. media.control. RecordControl.record	RecordControl.startRecord()	MultimediaRecord

Table 16 Individual permissions within Multimedia Record function group

9
JSR 75 – PIM and File Connection APIs

This chapter defines the JSR-75 APIs implementation requirements that shall replace the earlier implemented File Connection API requirement, except for the Recent Calls API that shall still be supported by *RecentCallRecord*, *RecentCallDialed* and *RecentCallReceived* classes.

Note: J2ME PIM and File Connection APIs should be implemented on J2ME platforms supporting CLDC 1.1 and MIDP 2.0 or higher.

PIM API

The primary goal of the PIM APIs is to provide access to Personal Information Management (PIM) data on J2ME enabled devices. PIM data is defined as information included in address book, calendar application, and to do list applications. To Do lists will be supported in the next versions of the API.

This chapter still details requirements for implementing Personal Information Management(PIM) and File Connection APIs specified in JSR -75 for J2ME enabled mobile devices.

Requirements

The implementation should include support of the following packages, classes, and interfaces with appropriate methods and fields of PIM API described in JSR-75 related to *javax.microedition.pim*:

- javax.microedition.pim.PIM;
- javax.microedition.pim.RepeatRule;
- javax.microedition.pim.PIMException;

- javax.microedition.pim.FieldEmptyException;
- javax.microedition.pim.FieldFullException;
- javax.microedition.pim.UnsupportedFieldException;
- javax.microedition.pim.PIMItem;
- javax.microedition.pim.Contact;
- javax.microedition.pim.Event;
- javax.microedition.pim.PIMList;
- javax.microedition.pim.ContactList;
- javax.microedition.pim.EventList.

The implementation should include support of the following packages, classes, and interfaces with appropriate methods and fields of FileConnection API described in JSR-75, related to *javax.microedition.io.file*:

- javax.microedition.io.file.ConnectionClosedException;
- javax.microedition.io.file.IllegalModeException;
- javax.microedition.io.file.FileConnection;

Security Requirements Personal information read/write permissions should be supported by the device's native system:

- *javax.microedition.pim.ContactList.read* - should enable reading the contact information available on the device (hereinafter just "contact read permission")
- *javax.microedition.pim.ContactList.write* - should enable updating the contact information available on the device (hereinafter just "contact write permission")
- *javax.microedition.pim.ContactList.write* - should enable updating the contact information available on the device (hereinafter just "contact write permission")
- *javax.microedition.pim.ContactList.write* - shall enable updating the contact information available on the device (hereinafter just "contact write permission")

The PIM permissions should be mapped to the function groups "User Data Read Capability" and "User Data Write Capability" depending on the read/write conditions. These two groups and the permissions are in the following Table 17:

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
User Data Read Capability	Always Ask, Ask Once Per App, Never Ask, No Access	No Access	Full Access	Full Access
User Data Write Capability	Always Ask, Ask Once Per App, Never Ask, No	No Access	Full Access	Full Access

	Access

Table 17 Permissions and Groups

The PIM permissions should prohibit granting to a MIDlet suite which does not request them explicitly in the attributes MIDlet -Permissions or MIDlet -Permissions – Opt.

The PIM package allows handling three types of lists: events, contacts and to do lists. Each one is stored in a specific database, respectively: event database, contact database and to do database. These databases have specific information of each list.

Contact List

The contact database contains data items representing personal contact information (like name, address, etc). The following features should be applied to the contact list:

- The implementation should provide support for ContactList type of PIM list as defined JSR-75.
- The implementation should provide a method to access an actual list of the PIM ContactList type.
- The implementation should provide interface to manipulate actual *ContactList* as specified in *ContactList* class section of JSR -75.
- The implementation should provide access to all available actual PIM lists for the ContactList list type.
- At least the following fields should be supported: UID, NICKNAME, TEL, EMAIL, FORMATTED_NAME, BIRTHDAY, ADDR_STREET, ADDR_LOCALI TY, ADDR_REGION, ADDR_POSTALCODE, ADDR_COUNTRY, ADDR_EXTRA, PHOTO_URL.
- At least the following attributes shall be supported: ATTR_MOBILE, ATTR_HOME, ATTR_WORK, ATTR_OTHER, ATTR_FAX, ATTR_PAGER.
- The location of the contact information (i.e. SIM card or Phone Memory) shall be defined by separate dedicated field content value.

Event List

The event database contains entries related on events (e.g. birthday). The following features should be applied to the contact list:

- The implementation should provide support for EventList type of PIM list as defined JSR -75.
- The implementation should provide a method to access an actual list of the PIM EventList type.
- The implementation should provide interface to manipulate actual EventList as specified in EventList class section of JSR-75.

- The implementation should provide access to all available actual PIM lists for the EventList list type.
- At least the following Event fields should be supported : SUMMARY, UID, END, START, ALARM.
- At least the following repeat rules fields should be supported: FREQUENCY, DAY_IN_WEEK, WEEK_IN_MONTH, DAY_IN_MONTH.
- At least one attribute should be supported: ATTR_NONE

To Do List

The To Do database contains entries to tasks that must be executed on determined data and times.

File Connection API

The primary goal of the FileConnection API is to provide access to file systems on devices and/or mounted removable memory cards supported by Motorola devices.

Requirements

File Connection API requirements will be replaced with the requirements below.

- The implementation should provide a security model for accessing the FileConnection APIs as defined in JSR-75.
- File Connection API should be accessible to manufacturer and operator domain MIDlets subject to security restrictions.
- Connection API should prohibit the modification or removing the files and directories marked with the system attribute.
- Call to System.getProperty with key microedition.io.file.FileConnection.version should return the implementation version number starting with 1.0.

Files read/write permissions should be supported by the device's native system:

- *javax.microedition.io.Connector.file.read* - should enable reading from the file system (hereinafter just "read permission").
- *javax.microedition.io.Connector.file.write* - should enable writing to the file system (hereinafter just "write permission").

The "read permission" and "write permission" should be mapped to the function groups "User Data Read Capability" and "User Data Write Capability" respectively. These two groups and permissions are in the Table 18:

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
User Data Read Capability	Always Ask, Ask Once Per App, Never Ask, No Access	No Access	Full Access	Full Access
User Data Write Capability	Always Ask, Ask Once Per App, Never Ask, No Access	No Access	Full Access	Full Access

Table 18 Groups and permissions for

The File Connection permissions should be prohibited for granting to a MIDlet suite which doesn't request them explicitly in the attributes MIDlet-Permissions or MIDlet-Permissions –Opt.

If the permission is not granted, a SecurityException shall be thrown by the following methods: *open, openDataInputStream, openInputStream, setFileConnection, listRoots, openDataOutputStream, openOutputStream; PIMList.items(all methods), PIMList.itemsByCategory, PIMList.addCategory, PIMList.deleteCategory, PIMList.renameCategory, PIMItem.commit, ContactList.removeContact, EventList.removeEvent, EventList.items*.

The following javax.*microedition.io.Connector* methods should check for the "read permission":

open("file://...");

open("file://...", Connector.READ);

open("file://...", Connector.READ_WRITE);

openDataInputStream();

openInputStream()

The following javax.microedition.io.file.FileConnection methods should check for the "read permission":

- *setFileConnection*, when instance opened with READ;
- *setFileConnection*, when instance opened with READ_WRITE.

The following javax.microedition.io.Connector methods should check for the "write permission":

open("file://...");

open("file://...", Connector.WRITE);

open("file://...", Connector.READ_WRITE);

openDataOutputStream();

openOutputStream();

openOutputStream(long byteOffset)

The following javax.microedition.io.file.FileConnection methods should check for the “write permission”:

- *setFileConnection*, when instance opened with WRITE*;*
- *setFileConnection,* when instance opened with READ_WRITE.

The bottom line prompt in the permission request dialog should include the name of the file or directory only for those protected API calls that have this information specified as a parameter.

The prompt prefix should be “<File Location>: <File Name>” for the following methods:

open; openDataInputStream;

openInputStream;

openDataOutputStream;

openOutputStream.

File Location would represent either:

- "Phone" (when the file is stored on the phone) .
- "Card" (when the file is stored on a MMC, SD, T-Flash or other card-related media).

10
JSR 184 – 3D API

Overview

JSR 184 Mobile 3D API defines an API for rendering three-dimensional (3D) graphics at interactive frame rates, including a scene graph structure and a corresponding file format for efficient management and deployment of 3D content. Typical applications that might make use of JSR 184 Mobile 3D API include games, map visualizations, user interface, animated messages, and screen savers. JSR 184 requires a J2ME device supporting MIDP 2.0 and CLDC 1.1 at a minimum.

Mobile 3D API

The Motorola V3x contains full implementation of JSR 184 Mobile 3D API (http://jcp.org/en/jsr/detail?id=184). The Motorola V3x has also implemented the following:

- Call to `System.getProperty` with key – microedition.m3g.version will return 1.0, otherwise null will be returned.
- Floating point format for input and output is the standard IEEE float having a 8-bit exponent and a 24-bit mantissa normalized to 1.0, 2.0.
- Implementation will ensure the Object3D instances will be kept reference to reduce overhead and possible inconsistency.
- Thread safety
- Necessary pixel format conversions for rendering output onto device
- Support at least 10 animation tracks to be associated with an Object 3D instance (including animation controller) subject to dynamic memory availability.

Mobile 3D API File Format Support

The Motorola V3x supports both M3G and PNG file formats for loading 3D content. The V3x supports the standard .m3g and .png extensions for its file formats. Mime type and not extension will be used for identifying file type. In the case that the Mime type is not available, M3G files will be identified using the file identifier and PNG files using signature.

Mobile 3D Graphics – M3G API

The M3G API lets you access the realtime 3D engine embedded on the device, to create console quality 3D applications, such as games and menu systems. The main benefits of the M3G engine are the following:

- the whole 3D scene can be stored in a very small file size (typically 50-150K), allowing you to create games and applications in under 256K;
- the application can change the properties (such as position, rotation, scale, color and textures) of objects in the scene based on user interaction with the device;
- the application can switch between cameras to get different views onto the scene;
- the rendered images have a very high photorealistic quality.

Typical M3G Application

An application consists of logic that uses the M3G, MIDP 2.0 and CDLC 1.1 classes. The application is compiled into a Java MIDlet that can be downloaded OTA or embedded on the target device. The MIDlet can also contain additional assets, such as one or more M3G files that define the 3D scene graph for the objects in the scene, images and sounds.

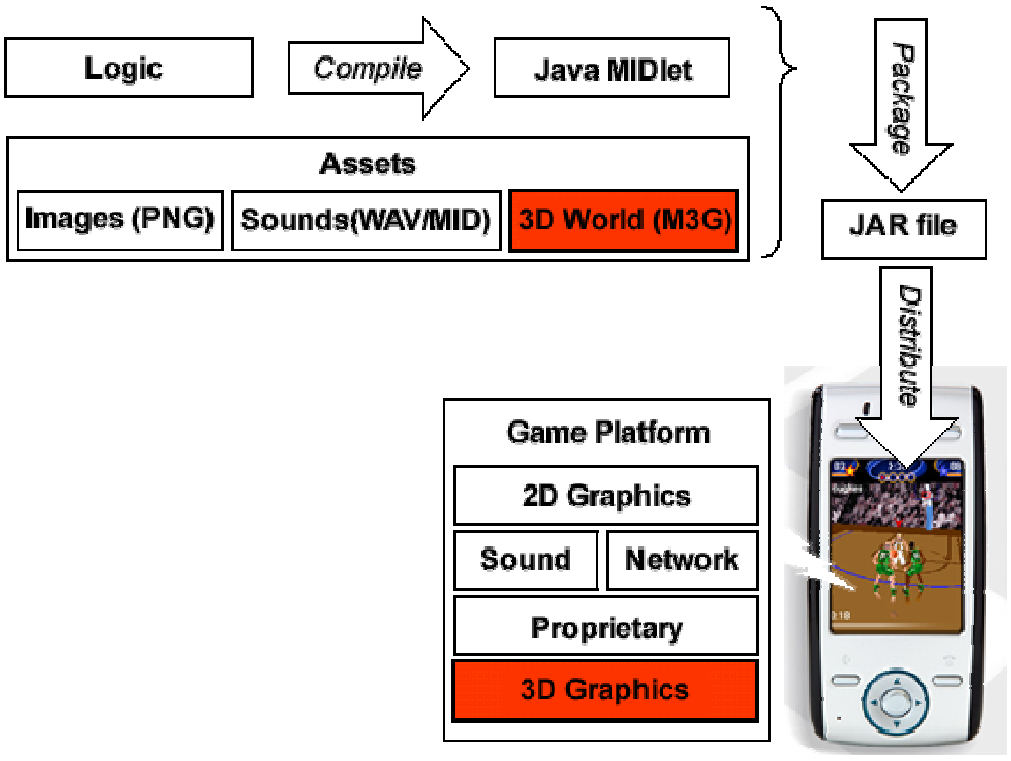

Figure 3 M3G Application Proccess

Most M3G applications use an M3G resource file that contains all the information required to define the 3D resources, such as objects, their appearance, lights, cameras and animations, in a scene graph. The file must be loaded into memory where object properties can be interrogated and altered using the M3G API. Alternatively all objects can be created from code, although this is likely to be slower and limits creativity for designers.

Simple MIDlets

The simplest application consists of an M3G file that is loaded into the application using the M3G Loader class, which is then passed to a Graphics3D object that renders the world to the Display.

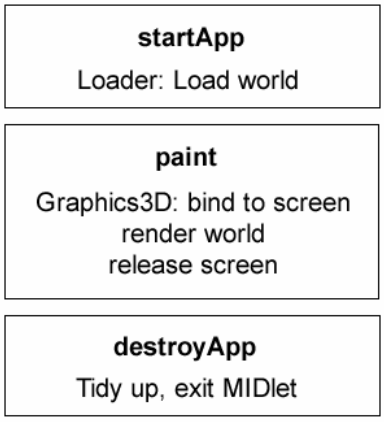

Figure 4 M3G Application Methods

The World object contains the objects that define a complete 3D scene - geometry, textures, lights, cameras, and animations. The World object mediates access to the objects within the world. It can be passed as a block to the renderer, the Graphics3D class.

The Loader object, populates a World by loading an M3G file from a URI or other asset source, such as a buffer of bytes in M3G format. The Loader is not restricted to loading just Worlds, each file can contain as little as a single object and multiple files can be merged together on the device, or you can put everything into a single file.

The rendering class Graphics3D (by analogy to the MIDP Graphics class) takes a whole scene (or part of a scene graph), and renders a view onto that scene using the current camera and lighting setup. This view can be to the screen, to a MIDP image, or to a texture in the scene for special effects. You can pass a whole world in one go (retained mode) or you can pass individual objects (immediate mode). There is only one Graphics3D object present at one time, so that hardware accelerators can be used.

The figure 5 shows the structure of a more typical MIDlet.

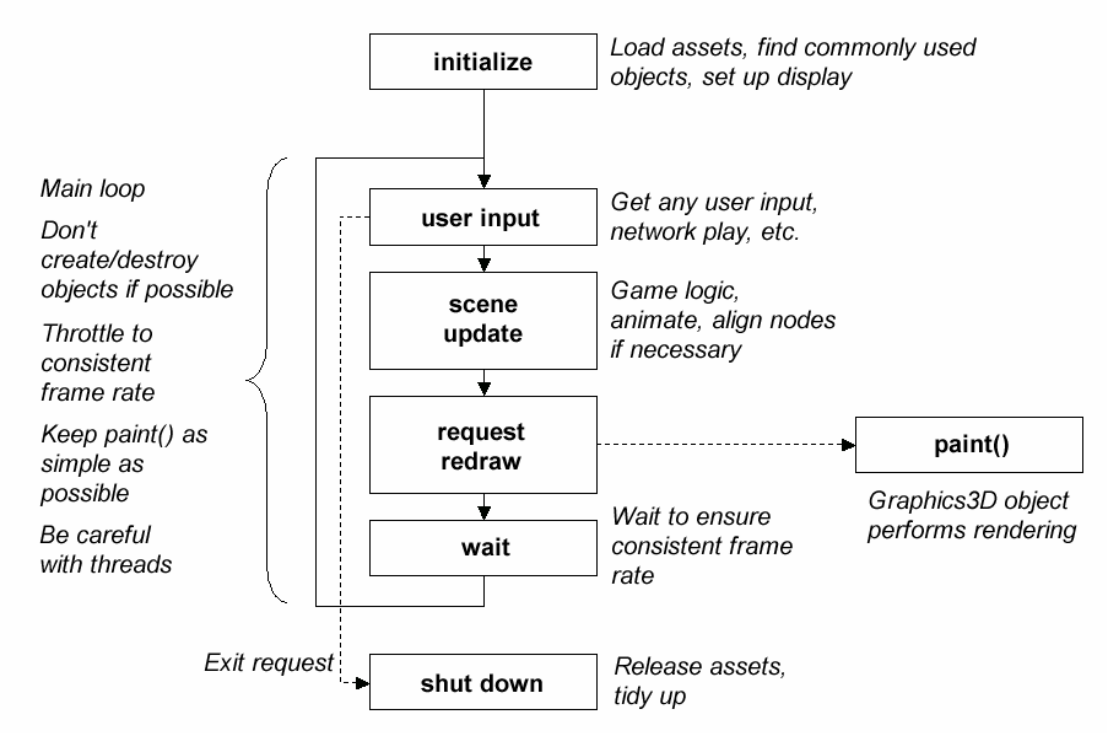

Figure 5 Typical MIDlet Structure

Initializing the world

The Loader class is used to initialize the world. It has two static methods: one takes in a byte array, while the other takes a named resource, such as a URI or an individual file in the JAR package.

The load methods return an array of Object3Ds that are the root level objects in the file.

The following example calls Loader.load() and passes it an M3G file from the JAR file using a property in the JAD file. Alternatively, you could specify a URI, for example:

```
Object3D[] roots =
Loader.load(http://www.example.com/m3g/simple.m3g)[0]);
```

The example assumes that there is only one root node in the scene, which will be the world object. If the M3G file has multiple root nodes the code must be changed to reflect this, but generally most M3G files have a single root node.

```
public void startApp() throws MIDletStateChangeException
{
      myDisplay.setCurrent(myCanvas);

      try
      {
            // Load a file.
```

```
            Objects3D[] roots = Loader.load(getAppProperty("Content-
1"));

            // Assume the world is the first root node loaded.
            myWorld = (World) roots[0];
      }
      catch(Exception e)
      {
            e.printStackTrace();
      }

      // Force a repaint so the update loop is started.
      myCanvas.repaint();
}
```

Code Sample 3 Initializing the world

Using the Graphics3D object

Using the Graphics3D is very straightforward. Get the Graphics3D instance, bind a target to it, render everything, and release the target.

```
public class myCanvas extends Canvas
{
      Graphics3D  myG3D = Graphics3D.getInstance();

      public void paint(Graphics g)
      {
            myG3D.bindTarget(g);
            try
            {
                  myG3D.render(myWorld);
            }
            finally
            {
                  myG3D.releaseTarget();
            }
```

```
        }
}
```

Code Sample 4 Using the Graphics3D object

The finally block makes sure that the target is released and the Graphics3D can be reused. The bindTarget call must be outside the try block, as it can throw exceptions that will cause releaseTarget to be called when a target has not been bound, and releaseTarget throwing an exception.

Interrogating and interacting with objects

The World object is a container that sits at the top of the hierarchy of objects that form the scene graph. You can find particular objects within the scene very simply by calling find() with an ID. find() returns a reference to the object which has been assigned that ID in the authoring tool (or manually assigned from code). This is important because it largely makes the application logic independent of the detailed structure of the scene.

```
final int   PERSON_OBJECT_ID = 339929883;
Node  personNode = (Node)theWorld.find(PERSON_OBJECT_ID);
```

If you need to find many objects, or you don't have a fixed ID, then you can follow the hierarchy explicitly using the Object3D.getReferences() or Group.getChild() methods.

```
static void traverseDescendants(Object3D obj)
{
      int numReferences = obj.getReferences(null);

      if (numReferences > 0)
      {
            Object3D[] objArray = new Object3D[numReferences];
            obj.getReferences(objArray);

            for (int i = 0; i < numReferences; i++)
                  traverseDescendants(objArray[i]);
      }
}
```

Once you have an object, most of the properties on it can be modified using the M3G API. For example, you can change the position, size, orientation, color, brightness, or whatever other attribute of the object is important. You can also create and delete objects and insert them into the world, or link parts of other M3G files into the scene graph.

Animations

As well as controlling objects from code, scene designers can specify how objects should move under certain circumstances, and store this movement in “canned” or block animation sequences that can be triggered from code. Many object properties are animatable, including position, scale, orientation, color and textures. Each of these properties can be attached to a sequence of keyframes using an AnimationTrack. The keyframe sequence can be looped, or just played once, and they can be interpolated in several ways (stepwise, linear, spline).

A coherent action typically requires the simultaneous animation of several properties on several objects, the tracks are grouped together using the AnimationController object. This allows the application to control a whole animation from one place.

All the currently active animatable properties can be updated by calling animate() on the World. (You can also call this on individual objects if you need more control). The current time is passed through to animate(), and is used to determine the interpolated value to assign to the properties.

The animate() method returns a validity value that indicates how long the current value of a property is valid. Generally this is 0 which means that the object is still being animated and the property value is no longer valid, or infinity when the object is in a static state and does not need to be updated. If nothing is happening in the scene, you do not have to continually redraw the screen, reducing the processor load and extending battery life. Similarly, simple scenes on powerful hardware may run very fast; by restricting the frame-rate to something reasonable, you can extend battery life and are more friendly to background processes.

The animation subsystem has no memory, so time is completely arbitrary. This means that there are no events reported (for example, animation finished). The application is responsible for specifying when the animation is active and from which position in the keyframe sequence the animated property is played.

Consider a world myWorld that contains an animation of 2000 ms, that you want to cycle. First you need to set up the active interval for the animation, and set the position of the sequence to the start. Then call World.animate() with the current world time:

```
anim.setActiveInterval(worldTime, worldTime+2000);
anim.setPosition(0, worldTime);


int validity = myWorld.animate(worldTime);
```

Authoring M3G files

You can create all your M3G content from code if necessary but this is likely to be very time consuming and does not allow 3D artists and scene designers to easily create and rework visually compelling content with complex animations. You can use professional, visual development tools such as Swerve™ Studio or Swerve™ M3G exporter from Superscape Group plc, which export content from 3ds max, the industry standard 3D animation tool, in fully compliant M3G format. For more information please visit http://www.superscape.com/.

11
JSR 185 - JTWI

JTWI specifies a set of services to develop highly portable, interoperable Java applications. JTWI reduces API fragmentation and broadens the number of applications for mobile phones.

Overview

Any Motorola device implementing JTWI will support the following minimum hardware requirements in addition to the minimum requirements specified in MIDP 2.0:

- At least a 125 x 125 pixels screen size as returned by full screen mode Canvas.getHeight () and Canvas.getWidth ()
- At least a color depth of 4096 colors (12-bit) as returned by Display.numColors ()
- Pixel shape of 1:1 ratio
- At least a Java Heap Size of 512 KB
- Sound mixer with at least 2 sounds
- At least a JAD file size of 5 KB
- At least a JAR file size of 64 KB
- At least a RMS data size of 30 KB

Any Motorola JTWI device will implement the following and pass the corresponding TCK:

- CLDC 1.0 or CLDC 1.1
- MIDP 2.0 (JSR 118)
- Wireless Messaging API 1.1 (JSR 120)
- Mobile Media API 1.1 (JSR 135)

CLDC related content for JTWI

JTWI is designed to be implemented on top of CLDC 1.0 or CLDC 1.1. The configuration provides the VM and the basic APIs of the application environment. If floating point capabilities are exposed to Java Applications, CLDC 1.1 will be implemented.

The following CLDC requirements will be supported:

- Minimum Application thread count will allow a MIDlet suite to create a minimum of 10 simultaneous running threads
- Minimum Clock Resolution – The method java.land.System.currentTimeMillis () will record the elapsed time in increments not to exceed 40 msec. At least 80% of test attemps will meet the time elapsed requirement to achieve acceptable conformance.
- Names for Encodings will support at least the preferred MIME name as defined by IANA (http://www.iana.org/assignments/character-sets) for the supported character encodings. If not preferred name has been defined, the registered name will be used (i.e UTF-16).
- Character Properties will provide support for character properties and case conversions for the characters in the Basic Latin and Latin-1 Supplement blocks of Unicode 3.0. Other Unicode character blocks will be supported as necessary.
- Unicode Version will support Unicode characters. Character information is based on the Unicode Standard version 3.0. Since the full character tables required for Unicode support can be excessively large for devices with tight memory budgets, by default, the character property and case conversion facilities in CLDC assume the presence of ISO Latin-1 range of characters only. Refer to JSR 185 for more information.
- Custom Time Zone Ids will permit use of custom time zones which adhere to the following time xone format:
    - General Time Zone: For time zones representing a GMT offset value, the following syntax is used:
        - Custom ID:
            - GMT Sign Hours: Minutes
            - GMT Sign Hours Minutes
            - GMT Sign Hours Hours
        - Sign: one of:
            - + -
        - Hours:
            - Digit
            - Digit Digit
        - Minutes:
            - Digit Digit

- Digit: one of:
    - 0 1 2 3 4 5 6 7 8 9

NOTE: Hours will be between 0 and 23, and minutes will be between 00 and 50. For example, GMT +10 and GMT +0010 equates to ten hours and ten minutes ahead of GMT.

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

MIDP 2.0 specific information for JTWI

MIDP 2.0 provides the library support for user interface, persistent storage, networking, security, and push functions. MIDP 2.0 contains a number of optional functions, some of which will be implemented as outlined below. The following JTWI requirements for MIDP 2.0 will be supported:

- Record Store Minimum will permit a MIDlet suite to create at least 5 independent RecordStores. This requirement does not intend to mandate that memory be reserved for these Record Stores, but it will be possible to create the RecordStores if the required memory is available.
- HTTP Support for Media Content will provide support for HTTP 1.1 for all supported media types. HTTP 1.1 conformance will match the MIDP 2.0 specification. See package.javax.microedition.io for specific requirements.
- JPEG for Image Objects – ISO/IEC JPEG together wil JFIF will be supported. The support for ISO/IEC JPEG only applies to baseline DCT, non-differential, Huffman coding, as defined in JSR 185 JTWI specification, symbol ‘SOF0’. This support extends to the class javax.microedition.lcdui.Image, including the methods outlined above. This mandate is voided in the event that the JPEG image format becomes encumbered with licensing requirements.
- Timer Resolution will 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. At least 80% of test attempts will meet the schedule resolution requirement to achieve acceptable conformance.

- Minimum Number of Timers will allow a MIDlet to create a minimum of 5 simultaneously running Timers. This requirement is independent of the minimum specified by the Minimum Application Thread Count.
- Bitmap Minimums will 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 format specification. For each of these color depths, as well as for JFIF image formats, a compliant implementation will support images up to 76800 total pixels.
- TextField and TextBox and Phonebook Coupling – when the center select key is pressed while in a TextBox or TextField and the constraint of the TextBox or TextField is TextField.PHONENUMBER, the names in the Phonebook will be displayed in the “Insert Phonenumber?” screen.
- Supported characters in TextField and TextBox – TextBox and TextField with input constraint TextField.ANY will support inputting all the characters listed in JSR 185.
- Supported characters in EMAILADDR and URL Fields – Class javax.microedition.lcdui.TextBox and javax.microedition.lcdui.TextField with either of the constraints TextField.EMAILADDR or TextField.URL will allow the same characters to be input as are allowed for input constraint TextField.ANY
- Push Registry Alarm Events will implement alarm-based push registry entries.
- Identification of JTWI via system property – to identify a compliant device and the implemented version of this specification, the value of the system property microedition.jtwi.version will be 1.0

Wireless Messaging API 1.1 (JSR 120) specific content for JTWI

WMA defines an API used to send and receive short messages. The API provides access to network-specific short message services such as GSM SMS or CDMA short messaging. JTWI will support the following as it is outlined in the JSR 120 chapter of this developer guide:

- Support for SMS in GSM devices
- Cell Broadcast Service in GSM devices
- SMS Push

Mobile Media API 1.1 (JSR 135) specific content for JTWI

The following will be supported for JTWI compliance:

- HTTP 1.1 Protocol will be supported for media file download for all supported media formats
- MIDI feature set specified in MMAPI (JSR 135) will be implemented. MIDI file playback will be supported.
- VolumeControl will be implemented and is required for controlling the colume of MIDI file playback.
- JPEG encoding in video snapshots will be supported if the handset supports the video feature set and video image capture.
- Tone sequence file format will be supported. Tone sequences provide an additional simple format for supporting the audio needs of many types of games and other applications.

MIDP 2.0 Security specific content for JTWI

- The Motorola V3x follows the security policy outlined in the Security chapter of this developer guide.

12
JSR 82 - Bluetooth API

Overview

JSR-82 covers the establishment of connections between devices for such applications as peer-to-peer gaming and Bluetooth pen use.

There are two new requirements from this API. The `javax.bluetooth` package is needed to establish general Bluetooth connections. The `javax.obex` package is needed to provide Object Exchange support over Bluetooth and other transports. Because OBEX is not limited to Bluetooth only, it resides as a separate package, but must be supported by this API.

JSR-82 Bluetooth API

The complete requirements are defined in Java™ APIs for Bluetooth™ Wireless Technology (JSR-82). The requirements listed here are a summary and specify how the API relates to the native Bluetooth implementation on the phone.

System Requirements

JSR-82 utilizes Bluetooth for data connections only. The following protocols must be supported:

- L2CAP
- RFCOMM
- SDP
- OBEX
    - OBEX is a separate API from the core Bluetooth API (`javax.bluetooth`) and is a part of the `javax.obex` package.

In addition, the following Bluetooth profiles must be supported:

- Generic Access Profile (GAP)
- Service Discovery Application Profile (SDAP)
- Serial Port Profile (SPP)
- Generic Object Exchange Profile (GOEP)

Bluetooth Control Center

The JSR-82 API requires that a Bluetooth Control Center (BCC) be in place to control the Bluetooth connection and be a repository for local device settings.

According to the API, the following are features the BCC must support:

- A list of remote Bluetooth devices (not necessarily in the vicinity) that are already known to the local Bluetooth device.
- A list of remote Bluetooth devices (not necessarily in the vicinity) that are trusted by the local Bluetooth device.
- A mechanism to bond two devices trying to connect for the first time.
- A mechanism to provide for authorization of connection requests.
- The base security settings of the local device, including the security modes defined in the Bluetooth specification.

Device Property Table

The Table 19 lists the Motorola Bluetooth device properties for current products. These device properties must be available to the MIDlet suite.

Device Property	Description
bluetooth.api.version	The version of the Java APIs for Bluetooth wireless technology that is supported. For this version it will be set to “1.0”.
bluetooth.l2cap.receiveMTU .max	The maximum ReceiveMTU size in bytes supported in L2CAP. The string will be in Base 10 digits, e.g., “672”. This value product dependent. The maximum value is 64 Kb.
bluetooth.connected.devices. max	The maximum number of connected devices supported (will include parked devices). The string will be in Base10 digits. This value is product dependent.
bluetooth.connected.inquiry	Is inquiry allowed during a connection? Valid values are either "true" or "false". This value is product dependent.
bluetooth.connected.page	Is paging allowed during a connection? Valid values are either "true" or "false". This value is product dependent.
bluetooth.connected.inquiry.	Is inquiry scanning allowed during connection? Valid values are either

scan	“true” or “false”. This value is product dependent.
bluetooth.connected.page.scan	Is page scanning allowed during connection? Valid values are either “true” or “false”. This value is product dependent.
bluetooth.master.switch	Is master/slave switch allowed? Valid values are either “true” or “false”. This value is product dependent.
bluetooth.sd.trans.max	Maximum number of concurrent service discovery transactions. The string will be in Base10 digits. This value is product dependent.
bluetooth.sd.attr.retrievable.max	Maximum number of service attributes to be retrieved per service record. The string will be in Base10 digits. This value is product dependent.

Table 19 Motorola Bluetooth device properties

Service Registration

Service Registration is the portion of the BCC that controls the Service Discovery Database (SDDB). The SDDB is a list of available services on the local device. Services registered in the SDDB by a MIDlet will be removed when the connection notifier is closed or when the MIDlet terminates.

The implementation must support run-before-connect services.

Connectable Mode

The following rules must be supported while the phone is in connectable mode:

Rules:

- In connectable mode, the Bluetooth device periodically listens for connection requests.
- The Bluetooth device will respond according to security settings and service availability for requested connection.

Non-Connectable Mode

In non-connectable mode, the Bluetooth device is neither discoverable nor connectable.

Device Management

Device Management describes the local settings involved that control how the local device responds to external requests.

Generic Access Profile (GAP)

These four GAP classes must be supported by the API:

- LocalDevice contains control settings of the local Bluetooth device. Settings can be read and changed.
- RemoteDevice contains information (i.e. Bluetooth address and friendly name) about a remote Bluetooth device.
- DeviceClass contains values of the device type and types of services the device supports.
- BluetoothStateException is an exception that is called when a request cannot be handled because of the device's state.

Security

Security must be set or controlled by the API. Parameters that are available to be set are:

- authentication
- encryption
- authorization
- master (for master/slave switch)

Communication

Communication covers establishing connections to other devices via specific Bluetooth profiles or protocols. Bluetooth connections established using this API are based on the following three protocols:

- RFCOMM
- L2CAP
- OBEX

Additionally, other profiles can be built upon these three basic protocols, but the profiles would have to be emulated by the MIDlet suite.

The implementation must support opening a connection with either a server connection URL or a client connection URL, with the default mode of READ_WRITE.

Serial Port Profile (SPP)

General Rules:

- SPP uses RFCOMM as its protocol.
- Only one RFCOMM session can exist between any pair of devices at one time.
- Negotiation of connection parameters and flow control between two Bluetooth devices must be handled automatically by the SPP connection implementation.
- A SPP server application must initialize the services it offers and register those services in the SDDB.

- Before an SPP client can establish a connection to an SPP service, it must discover that service via service discovery.
- A service discovery is not required if the SPP service had been discovered previously.

Object Exchange (OBEX)

OBEX is a protocol used for "pushing" and "pulling" objects (i.e. files or data) from one device to another. OBEX is not limited to Bluetooth only. OBEX can be used over Bluetooth, IrDA, and USB.

Rules:

- The following OBEX operations MUST be supported by the API:
  - CONNECT
  - PUT
  - GET
  - DISCONNECT
  - SETPATH
  - ABORT
  - CREATE-EMPTY
  - PUT-DELETE
- OBEX MUST support Bluetooth
- OBEX MAY support the following transports (where available)
  - IrDA
  - TCP/IP
- OBEX must support authentication.

Security Policy

Applications MUST be granted a permission to perform any requested operation using this API. The Table 20 assigns individual permissions to the function groups:

Bluetooth API JSR-82		
Permission	**Protocol**	**Function**
javax.bluetooth	Bluetooth	Data Networking

Table 20 Security Policy

External Events

The following interruptions must be handled by kvm and MIDlet suite.

Incoming Call

Rules:

Upon receiving an incoming call:

- The Bluetooth connection shall remain active when the MIDlet is suspended. The Bluetooth connection shall be terminated when the user Ends the MIDlet.

Incoming Message

Rules:

Upon receiving an incoming message:

- The Bluetooth connection shall remain active when the MIDlet is suspended. The Bluetooth connection shall be terminated when the user Ends the MIDlet.

Alarm & Datebook Behaviour

Rules:

The Alarm & Datebook behavior when a MIDlet is running:

- The Bluetooth connection shall remain active when the MIDlet is suspended. The Bluetooth connection shall be terminated when the user Ends the MIDlet.

Pressing of End Key

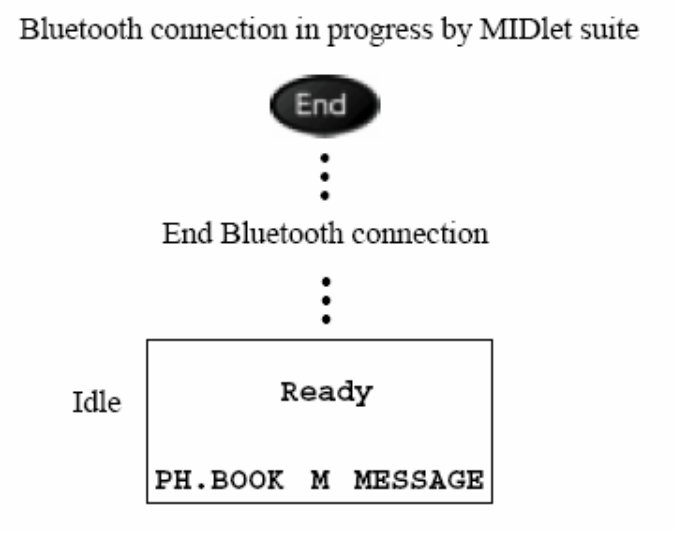

Figure 6 Pressing of End Key

Rules:

- Pressing the END key shall:
  - Terminate any ongoing Bluetooth connection.
    - If possible, notify other device that session will be disconnected.
- End MIDlet suite and kvm and return phone to Idle.

Hardware Requirements

Requires J2ME and Bluetooth wireless technology for the javax.bluetooth support.

Requires J2ME and at least one of the following: Bluetooth, IrDA, USB, or HTTP for javax.obex support.

Interoperability Requirements

SDK Developer/Style Guide Requirements:

The following table lists the suggested types of screens and text used for user feedback.

Examples of each screen type are provided below.

13
MIDP 2.0 Security Model

The following sections describe the MIDP 2.0 Default Security Model for the Motorola V3x handset. The chapter discusses the following topics:

- Untrusted MIDlet suites and domains
- Trusted MIDlet suites and domains
- Permissions
- Certificates

For a detailed MIDP 2.0 Security process diagram, refer to the Motocoder website (http://www.motocoder.com).

Refer to the table 21 for the default security feature/class support for MIDP 2.0:

Feature/Class	Implementation
All methods for the Certificate interface in the javax.microedition.pki package	Supported
All fields, constructors, methods, and inherited methods for the CertificateException class in the javax.microedition.pki package	Supported
MIDlet-Certificate attribute in the JAD	Supported
A MIDlet suite will be authenticated as stated in Trusted MIDletSuites using X.509 of MIDP 2.0 minus all root certificates processes and references	Supported
Verification of SHA-1 signatures with a MD5 message digest algorithm	Supported
Only one signature in the MIDlet-Jar-RSA-SHA1 attribute	Supported
All methods for the Certificate interface in the javax.microedition.pki package	Supported
All fields, constructors, methods, and inherited methods for the CertificateException class in the javax.microedition.pki package	Supported
Will preload two self authorizing Certificates	Supported
All constructors, methods, and inherited methods for the MIDletStateChangeException class in the javax.microedition.midlet	Supported

package	
All constructors and inherited methods for the MIDletStateChangeException class in the javax.microedition.midlet package	Supported

Table 21 MIDP 2.0 Feature/Class

Please note the domain configuration is selected upon agreement with the operator.

Untrusted MIDlet Suites

A MIDlet suite is untrusted when the origin or integrity of the JAR file cannot be trusted by the device.

The following are conditions of untrusted MIDlet suites:

- If errors occur in the process of verifying if a MIDlet suite is trusted, then the MIDlet suite will be rejected.
- Untrusted MIDlet suites will execute in the untrusted domain where access to protected APIs or functions is not allowed or allowed with explicit confirmation from the user.

Untrusted Domain

Any MIDlet suites that are unsigned will belong to the untrusted domain. Untrusted domains handsets will allow, without explicit confirmation, untrusted MIDlet suites access to the following APIs:

- `javax.microedition.rms` – RMS APIs
- `javax.microedition.midlet` – MIDlet Lifecycle APIs
- `javax.microedition.lcdui` – User Interface APIs
- `javax.microedition.lcdui.game` – Gaming APIs
- `javax.microedition.media` – Multimedia APIs for sound playback
- `javax.microedition.media.control` – Multimedia APIs for sound playback

The untrusted domain will allow, with explicit user confirmation, untrusted MIDlet suites access to the following protected APIs or functions:

- `javax.microedition.io.HttpConnection` – HTTP protocol
- `javax.microedition.io.HttpsConnection` – HTTPS protocol

Trusted MIDlet Suites

Trusted MIDlet suites are MIDlet suites in which the integrity of the JAR file can be authenticated and trusted by the device, and bound to a protection domain. The Motorola V3x will use x.509PKI for signing and verifying trusted MIDlet suites.

Security for trusted MIDlet suites will utilize protection domains. Protection domains define permissions that will be granted to the MIDlet suite in that particular domain. A MIDlet suite will belong to one protection domain and its defined permissible actions. For implementation on the Motorola V3x, the following protection domains should exist:

- Manufacturer – permissions will be marked as "Allowed" (Full Access). Downloaded and authenticated manufacturer MIDlet suites will perform consistently with MIDlet suites pre-installed by the manufacturer.
- Operator – permissions will be marked as "Allowed" (Full Access). Downloaded and authenticated operator MIDlet suites will perform consistently with other MIDlet suites installed by the operator.
- 3rd Party – permissions will be marked as "User". User interaction is required for permission to be granted. MIDlets do not need to be aware of the security policy except for security exceptions that will occur when accessing APIs.
- Untrusted – all MIDlet suites that are unsigned will belong to this domain.

Permissions within the above domains will authorize access to the protected APIs or functions. These domains will consist of a set of "Allowed" and "User" permissions that will be granted to the MIDlet suite.

Permission Types concerning the Handset

A protection domain will consist of a set of permissions. Each permission will be "Allowed" or "User", not both. The following is the description of these sets of permissions as they relate to the handset:

"Allowed" (Full Access) permissions are any permissions that explicitly allow access to a given protected API or function from a protected domain. Allowed permissions will not require any user interaction.

"User" permissions are any permissions that require a prompt to be given to the user and explicit user confirmation in order to allow the MIDlet suite access to the protected API or function.

User Permission Interaction Mode

User permission for the Motorola V3x handsets is designed to allow the user the ability to either deny or grant access to the protected API or function using the following interaction modes (bolded term(s) is prompt displayed to the user):

blanket – grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is uninstalled or the permission is changed by the user. (Never Ask)

session – grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is terminated. This mode will prompt the user on or before the final invocation of the protected API or function. (Ask Once Per App)

oneshot – will prompt the user each time the protected API or function is requested by the MIDlet suite. (Always Ask)

No – will not allow the MIDlet suite access to the requested API or function that is protected. (No Access)

The prompt No, Ask Later will be displayed during runtime dialogs and will enable the user to not allow the protected function to be accessed this instance, but to ask the user again the next time the protected function is called.

User permission interaction modes will be determined by the security policy and device implementation. User permission will have a default interaction mode and a set of other available interaction modes. The user should be presented with a choice of available interaction modes, including the ability to deny access to the protected API or function. The user will make their decision based on the user-friendly description of the requested permissions provided for them.

The Permissions menu allows the user to configure permission settings for each MIDlet when the VM is not running. This menu is synchronized with available run-time options.

Implementation based on Recommended Security Policy

The required trust model, the supported domain, and their corresponding structure will be contained in the default security policy for Motorola's implementation for MIDP 2.0. Permissions will be defined for MIDlets relating to their domain. User permission types, as well as user prompts and notifications, will also be defined.

Trusted 3rd Party Domain

A trusted third party protection domain root certificate is used to verify third party MIDlet suites. These root certificates will be mapped to a location on the handset that cannot be modified by the user. The storage of trusted third party protection domain root certificates and operator protection domain root certificates in the handset is limited to 12 certificates.

If a certificate is not available on the handset, the third party protection domain root certificates will be disabled. The user will have the ability to disable root certificates through the browser menu and will be prompted to warn them of the consequences of

disabling root certificates. These third party root certificates will not be used to verify downloaded MIDlet suites.

The user will be able to enable any disabled trusted third party protection domain root certificates. If disabled, the third party domain will no longer be associated with this certificate. Permissions for trusted third party domain will be “User” permissions; specifically user interaction is required in order for permissions to be granted.

The table 22 shows the specific wording to be used in the first line of the above prompt:

Protected Functionality	Top Line of Prompt	Right Softkey
Data Network	Use data network?	OK
Messaging	Use messaging?	OK
App Auto-Start	Launch <MIDlet names>?	OK
Connectivity Options	Make a local connection?	OK
User Data Read Capability	Read phonebook data?	OK
User Data Write Capability	Modify phonebook data?	OK
App Data Sharing	Share data between apps?	OK

Table 22 Trusted 3rd Party Domain

The radio button messages will appear as follows and mapped to the permission types as shown in the table 23:

MIDP 2.0 Permission Types	Runtime Dialogs	UI Permission Prompts
Oneshot	Yes, Always Ask	Always Ask
Session	Yes, Ask Once	Ask Once per App
Blanket	Yes, Always Grant Access	Never Ask
no access	No, Never Grant Access	No, Access

Table 23 MIDP 2.0 Permission Types

The above runtime dialog prompts will not be displayed when the protected function is set to “Allowed” (or full access), or if that permission type is an option for that protected function according to the security policy table flexed in the handset.

Security Policy for Protection Domains

The table 24 lists the security policy by function groups for each domain. Under each domain are the settings allowed for that function within the given domain, while the bolded setting is the default setting. The Function Group is what will be displayed to the user when access is requested and when modifying the permissions in the menu. The default setting is the setting that is effective at the time the MIDlet suite is first invoked and remains in effect until the user changes it.

Permissions can be implicitly granted or not granted to a MIDlet based on the configuration of the domain the MIDlet is bound to. Specific permissions cannot be defined for this closed class. A MIDlet has either been developed or not been developed to utilize this capability. The other settings are options the user is able to change from the default setting.

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
Data Network	**Ask Once Per App**, Always Ask, Never Ask, No Access	**Always Ask**, Ask Once Per App, No Access	Full Access	Full Access
Messaging	**Always Ask**, No Access	**Always Ask**, No Access	Full Access	Full Access
App Auto-Start	**Ask Once Per App**, Always Ask, Never Ask, No Access	**Ask Once Per App**, Always Ask, No Access	Full Access	Full Access
Connectivity Options	**Ask Once Per App**, Always Ask, Never Ask, No Access	**Ask Once Per App**, Always Ask, Never Ask, No Access	Full Access	Full Access
User Data Read Capability	**Always Ask**, Ask Once Per App, Never Ask, No Access	**No Access**	Full Access	Full Access
User Data Write Capability	**Always Ask**, Ask Once Per App, Never Ask, No Access	**No Access**	Full Access	Full Access
Multimedia Recording	**Ask Once Per App**, Always Ask, Never Ask, No Access	**No Access**	Full Access	Full Access

Table 24 Security Policy for Protection Domains

The table 25 shows individual permissions assigned to the function groups shown in the table above.

MIDP 2.0 Specific Functions		
Permission	**Protocol**	**Function Group**
javax.microedition.io.Connector.http	http	Data Network
javax.microedition.io.Connector.https	https	Data Network
javax.microedition.io.Connector.datagram	Datagram	Data Network
javax.microedition.io.Connector.datagramreceiver	datagram server (w/o host)	Data Network
javax.microedition.io.Connector.socket	Socket	Data Network
javax.microedition.io.Connector.serversocket	server socket (w/ o host)	Data Network
javax.microedition.io.Connector.ssl	Ssl	Data Network
javax.microedition.io.Connector.comm	Comm.	Connectivity Options
javax.microedition.io.PushRegistry	All	App Auto-Start
Wireless Messaging API - JSR 120		
javax.wireless.messaging.sms.send		Messaging
javax.wireless.messaging.sms.receive		Messaging
javax.microedition.io.Connector.sms		Messaging
javax.wireless.messaging.cbs.receive		Messaging
Multimedia Recording		
javax.microedition.media.RecordControl.startRecord	RecordControl.startRecord ( )	Multimedia Recording

Table 25 MIDP 2.0 Specific Functions

Each phone call or messaging action will present the user with the destination phone number before the user approves the action. The handset will ensure that I/O access from the Mobile Media API follows the same security requirements as the Generic Connection Framework.

Displaying of Permissions to the User

Permissions will be divided into function groups and two high-level categories, with the function groups being displayed to the user. These two categories are Network/Cost related and User/Privacy related.

The Network/Cost related category will include net access, messaging, application auto invocation, and local connectivity function groups.

The user/privacy related category will include multimedia recording, read user data access, and the write user data access function groups. These function groups will be displayed in the settings of the MIDlet suite.

Only 3rd party and untrusted permissions can be modified or accessed by the user. Operator and manufacturer permissions will be displayed for each MIDlet suite, but cannot be modified by the user.

Trusted MIDlet Suites Using x.509 PKI

Using the x.509 PKI (Public Key Infrastructure) mechanism, the handset will be able to verify the signer of the MIDlet suite and bind it to a protection domain which will allow the MIDlet suite access to the protected API or function. Once the MIDlet suite is bound to a protection domain, it will use the permission defined in the protection domain to grant the MIDlet suite access to the defined protected APIs or functions.

The MIDlet suite is protected by signing the JAR file. The signature and certificates are added to the application descriptor (JAD) as attributes and will be used by the handset to verify the signature. Authentication is complete when the handset uses the root certificate (found on the handset) to bind the MIDlet suite to a protection domain (found on the handset).

Signing a MIDlet Suite

The default security model involves the MIDlet suite, the signer, and public key certificates. A set of root certificates are used to verify certificates generated by the signer. Specially designed certificates for code signing can be obtained from the manufacturer, operator, or certificate authority. Only root certificates stored on the handset will be supported by the Motorola V3x handset.

Signer of MIDlet Suites

The signer of a MIDlet suite can be the developer or an outside party that is responsible for distributing, supporting, or the billing of the MIDlet suite. The signer will have a public key infrastructure and the certificate will be validated to one of the protection domain root certificates on the handset. The public key is used to verify the signature of JAR on the

MIDlet suite, while the public key is provided as a x.509 certificate included in the application descriptor (JAD).

MIDlet Attributes Used in Signing MIDlet Suites

Attributes defined within the manifest of the JAR are protected by the signature. Attributes defined within the JAD are not protected or secured. Attributes that appear in the manifest (JAR file) will not be overridden by a different value in the JAD for all trusted MIDlets. If a MIDlet suite is to be trusted, the value in the JAD will equal the value of the corresponding attribute in the manifest (JAR file), if not, the MIDlet suite will not be installed.

The attributes MIDlet-Permissions (-OPT) are ignored for unsigned MIDlet suites. The untrusted domain policy is consistently applied to the untrusted applications. It is legal for these attributes to exist only in JAD, only in the manifest, or in both locations. If these attributes are in both the JAD and the manifest, they will be identical. If the permissions requested in the HAD are different than those requested in the manifest, the installation will be rejected.

Methods:

1. MIDlet.getAppProperty will return the attribute value from the manifest (JAR) if one id defined. If an attribute value is not defined, the attribute value will return from the application descriptor (JAD) if present.

Creating the Signing Certificate

The signer of the certificate will be made aware of the authorization policy for the handset and contact the appropriate certificate authority. The signer can then send its distinguished name (DN) and public key in the form of a certificate request to the certificate authority used by the handset. The CA will create a x.509 (version 3) certificate and return to the signer. If multiple CAs are used, all signer certificates in the JAD will have the same public key.

Inserting Certificates into JAD

When inserting a certificate into a JAD, the certificate path includes the signer certificate and any necessary certificates while omitting the root certificate. Root certificates will be found on the device only.

Each certificate is encoded using base 64 without line breaks, and inserted into the application descriptor as outlined below per MIDP 2.0.

```
MIDlet-Certificate-<n>-<m>: <base64 encoding of a
certificate>
```

Note the following:

<n>:= a number equal to 1 for first certification path in the descriptor, or 1 greater than the previous number for additional certification paths. This defines the sequence in which the certificates are tested to see if the corresponding root certificate is on the device.

<m>:= a number equal to 1 for the signer's certificate in a certification path or 1 greater than the previous number for any subsequent intermediate certificates.

Creating the RSA SHA-1 signature of the JAR

The signature of the JAR is created with the signer's private key according to the EMSA-PKCS1 –v1_5 encoding method of PKCS #1 version 2.0 standard from RFC 2437. The signature is base64 encoded and formatted as a single MIDlet-Jar-RSA-SHA1 attribute without line breaks and inserted into the JAD.

It will be noted that the signer of the MIDlet suite is responsible to its protection domain root certificate owner for protecting the domain's APIs and protected functions; therefore, the signer will check the MIDlet suite before signing it. Protection domain root certificate owners can delegate signing MIDlet suites to a third party and in some instances, the author of the MIDlet.

Authenticating a MIDlet Suite

When a MIDlet suite is downloaded, the handset will check the JAD attribute MIDlet-Jar-RSA-SHA1. If this attribute is present, the JAR will be authenticated by verifying the signer certificates and JAR signature as described. MIDlet suites with application descriptors that do not have the attributes previously stated will be installed and invoked as untrusted. For additional information, refer to the MIDP 2.0 specification.

Verifying the Signer Certificate

The signer certificate will be found in the application descriptor of the MIDlet suite. The process for verifying a Signer Certificate is outlined in the steps below:

1. Get the certification path for the signer certificate from the JAD attributes MIDlet-Certificate-1<m>, where <m> starts a 1 and is incremented by 1 until there is no attribute with this name. The value of each attribute is abase64 encoded certificate that will need to be decoded and parsed.
2. Validate the certification path using the basic validation process as described in RFC2459 using the protection domains as the source of the protection domain root certificates.
3. Bind the MIDlet suite to the corresponding protection domain that contains the protection domain root certificate that validated the first chain from signer to root.
4. Begin installation of MIDlet suite.

5. If attribute MIDlet-Certificate-<n>-<m> with <n> being greater than 1 are present and full certification path could not be established after verifying MIDlet-Certificate-<1>-<m> certificates, then repeat step 1 through 3 for the value <n> greater by 1 than the previous value.

The Table 26 describes actions performed upon completion of signer certificate verification:

Result	Action
Attempted to validate <n> paths. No public keys of the issuer for the certificate can be found, or none of the certificate paths can be validated.	Authentication fails, JAR installation is not allowed.
More than one full certification path is established and validated.	Implementation proceeds with the signature verification using the first successfully verified certificate path for authentication and authorization.
Only one certification path established and validated.	Implementation proceeds with the signature verification.

Table 26 Actions performed of signer certificate verification

Verifying the MIDlet Suite JAR

The following are the steps taken to verify the MIDlet suite JAR:

1. Get the public key from the verified signer certificate.
2. Get the MIDlet-JAR-RSA-SHA1 attribute from the JAD.
3. Decode the attribute value from base64 yielding a PKCS #1 signature, and refer to RFC 2437 for more detail.
4. Use the signer's public key, signature, and SHA-1 digest of JAR to verify the signature. If the signature verification fails, reject the JAD and MIDlet suite. The MIDlet suite will not be installed or allow MIDlets from the MIDlet suite to be invoked as shown in the following table.
5. Once the certificate, signature, and JAR have been verified, the MIDlet suite is known to be trusted and will be installed (authentication process will be performed during installation).

The Table 27 is a summary of MIDlet suite verification including dialog prompts:

Initial State	Verification Result
JAD not present, JAR downloaded	Authentication can not be performed, will install JAR. MIDlet suite is treated as untrusted. The following error prompt will be shown,

	"Application installed, but may have limited functionality."
JAD present but is JAR is unsigned	Authentication can not be performed, will install JAR. MIDlet suite is treated as untrusted. The following error prompt will be shown, "Application installed, but may have limited functionality."
JAR signed but no root certificate present in the keystore to validate the certificate chain	Authentication can not be performed. JAR installation will not be allowed. The following error prompt will be shown, "Root certificate missing. Application not installed."
JAR signed, a certificate on the path is expired	Authentication can not be completed. JAR installation will not be allowed. The following error prompt will be shown, "Expired Certificate. Application not installed."
JAR signed, a certificate rejected for reasons other than expiration	JAD rejected, JAR installation will not be allowed. The following error prompt will be shown, "Authentication Error. Application not installed."
JAR signed, certificate path validated but signature verification fails	JAD rejected, JAR installation will not be allowed. The following error prompt will be shown, "Authentication Error. Application not installed."
Parsing of security attributes in JAD fails	JAD rejected, JAR installation will not be allowed. The following error prompt will be shown, "Failed Invalid File."
JAR signed, certificate path validated, signature verified	JAR will be installed. The following prompt will be shown, "Installed."

Table 27 Summary of MIDlet suite verification

Carrier Specific Security Model

The MIDP 2.0 security model will vary based on carrier requests. Contact the carrier for specifics.

Bound Certificates

Bound certificates enable an efficient process to aid developers in the MIDlet development and testing phase when working with signed applications. Currently the delay for the developer occurs because specific flex files need to be created for each developer and for each domain being tested.

By implementing Bound certificates, the process of creating and supporting developer-specific flex files can be eliminated, which in turn simplifies the developer's environment, avoiding dependency from the flex tools. Bound certificates will take advantage of the High Assurance Boot system implemented at Motorola. The main difference becomes relevant during the creation of the signing certificate for the developer. Below are the steps necessary for the developer to follow:

- The MIDlet developer generates a signing key that contains public and private keys.
- The developer will send the CSR, containing the developer's public portion of the signing key, and the serial number(s) of the handset(s) the developer is using for testing, and the intended protected domains the MIDlet will be signed against to the Motorola Java Signing Center.
- The Signing Center constructs a developer certificate that includes the public key and a tag, that denotes this is a bound certificate.
- This bound certificate has the serial number for the unit appended to the certificate format and the resulting file is signed using the PCS Java CA.
- When the phone starts to load this type of certificate, it will identify the bound tag and pull the electronic number from the processor and use it to validate the signature of that certificate. Once this validation takes place, then the certificate will be used to validate the signature of the JAD file and if it passes, then it will install the JAR file on the product.

This implementation incorporates information about the target domain into the bound certificate used for signing. This information should be submitted along with developer's CSR and bound tag(s) of the target device(s). If Java security manager has this information in runtime, it will be able to decide what domain to use for binding.

Following are the requirements concerning these bound certificates:

- An X.509 bound certificate shall support at least 10 serial numbers supplied in the special bound tag extension.
- A bound MIDlet shall be successfully installed on the target device, if at least one of serial numbers supplied in the bound certificate coincides with the processor's serial number retrieved from the target device.
- A bound MIDlet shall not be installed if bound tag verification fails.
- A bound MIDlet, after successful bound tag check, shall be successfully mapped to the hardcoded, SRP* compliant manufacturer domain, if the bound certificate includes information about target developer's domain where all permissions have type allowed.
- A bound MIDlet, after successful bound tag check, shall be successfully mapped to the hardcoded, SRP* compliant 3rd party domain, if the bound certificate includes information about target developer's domain where all permissions have type user.
- A bound MIDlet, after successful bound tag check, shall be mapped to a domain in accordance with flexed policy file, if the bound certificate either doesn't include any information about target developer's domain or includes information about domain unknown to the device. The

MIDlet shall not be installed if the domain policy flexed on the target device doesn't include an appropriate domain.

* SRP - MIDP 2.0 Security Recommended Practice for GSM/UMTS compliant devices.

14
Prevent Downloading of Large Java MIDlets

Overview

This feature makes flexible way of preventing the large JAR files OTA download. The current functionality is as follows:

- The user is able to download any JAR file independently from its file size via the WAP browser. In some cases the MIDlet can not be executed properly due to the limited heap memory size of a Java enabled phone. In this case the user is charged for the data transfer (not for the event) and in some cases this charge will be higher than the cost of MIDlet itself.

In order to let the different operators utilize this feature there shall be ability to use flex database element to limit the maximum size of the JAR file. The appropriate notice shall be displayed to inform the user that the maximum JAR file size is exceeded and downloading is rejected. The maximum JAR file size value shall come directly from a customer requirement or product team.

There may be 2 types of MIDlets download:

- JAR-only file download
- JAR/JAD file download

This feature does not consider JAR-only download. The behavior in this case is specified by "J2ME Download MIDlet Through Browser" feature.

- The system shall support setting the maximum JAR file size for downloading.
- This feature shall apply only to JAR accompanied by JAD files download.
- The maximum JAR file size for the download shall be stored in the flex database.
- The default size value shall be set to maximum available value for the flex database element.

- The system shall only download JAR files that are less than or equal to the maximum size specified in flex database.
- The size of JAR accompanied by JAD files download shall be controlled by the flex element.

Notification

When the JAR file size exceeds the maximum value set, then the notice shall appear to inform the user that the JAR file download was aborted.

When the JAR file size exceeds the maximum value set and the downloading was aborted, then the following notification report shall be sent to the server: "901 Insufficient Memory".

When the size of the JAR file exceeds the maximum value, the JAVA AMS shall initiate the transient notice.

Backward Compatibility/Flexing

The maximum JAR file size shall be configured as per operator requirement on product by product basis.

15
Launch native streaming video client from Java application

Overview

This feature delivers a capability for a Java application (MIDlet) to launch a native video streaming client on the handset. Possible use case may be for the user to use a MIDlet to access a streaming video programming channel on the network, select a channel to watch, and the MIDlet launching the native streaming video application to deliver the picture to user.

New Implementation

- The system SHALL provide support for a MIDlet to launch the streaming video native application.
- User SHALL be prompt to approve the connection to the network, using the existing prompt.
- The streaming video application SHALL start streaming from the URL passed by the MIDlet as soon as it up and running, without user intervention.
- The streaming client SHALL use the dedicated streaming APN configuration to access the content.
- The MIDlet will not pass APN information to the streaming client.
- The MIDlet SHALL obtain the type of radio network coverage from the handset using a dedicated API method.
- The method to obtain the type of radio network coverage SHALL return one of 3 values to the MIDlet:
    - GPRS
    - EDGE
    - UMTS

16
JSR 139 – CLDC 1.1

JSR 139

CLDC 1.1 is an incremental release of CLDC version 1.0. CLDC 1.1 is fully backwards compatible with CLDC 1.0. Implementation of CLDC 1.1 supports the following:

- Floating Point
  - Data Types float and double
  - All floating point byte codes
  - New Data Type classes Float and Double
  - Library classes to handle floating point values
- Weak reference
- Classes Calender, Date and TimeZone are J2SE compliant
- Thread objects to be compliant with J2SE.

The support of thread objects to be compliant with J2SE requires the addition of `Thread.getName` and a few new contructors. The following table lists the additional classes, fields, and methods supported for CLDC 1.1 compliance:

	Classes	Additional Fields/Methods	Comments
System Classes	Java.lang.Thread	Thread (Runnable target, String name)	Allocates a new Thread object with the given target and name.
		Thread (String name)	Allocates a new Thread object with the given name
		String getName ()	Returns this thread's name
		Void interrupt ()	Interrupts this thread
	Java.lang.String	Boolean equalIgnoreCase (String anotherString)	Compares this string to another String, ignoring case considerations

		String intern ()	Returns a canonical representation for the string object
		Static String valueOf (float f)	Returns the string representation of the float argument
		Static String valueOf (double d)	Returns the string representation of the double argument
Data Type Classes	Java.lang.Float		New Class: Refer to CLDC Spec for more details
	Java.lang.Double		New Class: Refer to CLDC Spec for more details
Calender and Time Classes	Java.util.Calender	Protected int [ ] fields	The field values for the currently set time for this calendar
		Protected boolean { } is set	The flags which tell if a specified time field for the calendar is set
		Protected long time	The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT
		Protected abstract void ComputeFields	Converts the current millisecond time value to field values in fields [ ]
		Protected abstract void ComputeTime	Converts the current field values in fields [ ] to the millisecond time value time
	Java.lang.Date	String toString ()	Converts this date object to a String of the form: Dow mon dd hh:mm:ss zzz yyyy
Exception and Error Classes	Java.lang.NoClass DefFoundError		New Class: Refer to CLDC Spec for more details
Weak References	Java.lang.ref.Refer ence		New Class: Refer to CLDC Spec for more details
	Java.lang.ref.Weak Reference		New Class: Refer to CLDC Spec for more details
Additional Utility Classes	Java.util.Random	Double nextDouble ()	Returns the nextpseudorandom, uniformly distributed double value between 0.0 and 1.0 from the random number generator's sequence
		Float nextFloat ()	Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from the random number generator's

			sequence
		Int nextInt (int n)	Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence
	Java.lang.Math	Static double E	The double value that is closer than any other to e, the base of the natural logarithms
		Static double PI	The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter
		Static double abs (double a)	Returns the absolute value of a double value
		Static float abs (float a)	Returns the absolute value of a double value
		Static double ceil (double a)	Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer
		Static double cos (double a)	Returns the trigonometric cosine of an angle
		Static double floor (double a)	Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.
		Static double max (double a, double b)	Returns the greater of two double values
		Static float max (float a, float b)	Returns the greater of two float values
		Static double min (float a, float b)	Returns the smaller of two double values
		Static float min (float a, float b)	Returns the smaller of two float values
		Static double sin (double a)	Returns the trigonometric sine of an angle
		Static double sqrt (double a)	Returns the correctly rounded positive square root of a double value

		Static double tan (double a)	Returns the trigonometric tangent of angle
		Static double todegrees (double angrad)	Converts an angle measured in radians to the equivalent angle measured in degrees
		Static double toradians (double angdeg)	Converts an angle measured in degrees to the equivalent angle measured in radians

Table 28 Additional classes, fields, and methods supported for CLDC 1.1 compliance

Appendix A: Key Mapping

Key Mapping for the V3x

Table 29 identifies key names and corresponding Java assignments. All other keys are not processed by Java.

Key	Assignment
0	NUM0
1	NUM1
2	NUM2
3	NUM3
4	NUM4
5	SELECT, followed by NUM5
6	NUM6
7	NUM7
8	NUM8
9	NUM9
STAR (*)	ASTERISK
POUND (#)	POUND
JOYSTICK LEFT	LEFT
JOYSTICK RIGHT	RIGHT
JOYSTICK UP	UP
JOYSTICK DOWN	DOWN
SCROLL UP	UP
SCROLL DOWN	DOWN
SOFTKEY 1	SOFT1
SOFTKEY 2	SOFT2
MENU	SOFT3 (MENU)
SEND	SELECT Also, call placed if pressed on lcdui.TextField or lcdui.TextBox with PHONENUMBER constraint set.
CENTER SELECT	SELECT
END	Handled according to Motorola specification: Pause/End/Resume/Background menu invoked.

Table 29 Key Mapping

Table 30 identifies keys that will be assigned to game actions defined in GameCanvas class of MIDP 2.0.

Action	First Set	Second Set	Third Set	Non-simultaneous keys
Left	Nav (LEFT)	4		
Right	Nav (RIGHT)	6		
Up	Nav (UP)	2		
Down	Nav (DOWN)	8		
Game_A			0	
Game_B				1
Game_C				3
Game_D				5
Game_Fire	9	7	#	

Table 30 GameCanvas class of MIDP 2.0

Appendix B: Memory Management Calculation

Available Memory

The available memory on the V3x is the following:

- 64MB shared memory for MIDlet storage
- 800 Kb Heap size

Appendix C: FAQ

Online FAQ

The MOTOCODER developer program is online and able to provide access to Frequently Asked Questions around enabling technologies on Motorola products.

Access to dynamic content based on questions from the Motorola J2ME developer community is available at the URL listed below.

http://www.motocoder.com

Appendix F: Spec Sheet V3x

V3x Spec Sheet

Listed below are the spec sheets for the V3x. The spec sheets contain information regarding the following areas:

- Technical Specifications
- Key Features
- J2ME Information
- Motorola Developer Information
- Tools
- Other Related Information

MOTOROLA
intelligence everywhere™

Motorola V3x

Developer Reference Sheet

Technical Specifications

Band/Frequency	WCDMA 2100 GSM 900/1800/1900 GPRS 900/1800/1900
Region	Global
Technology	WAP 2.0, J2ME, SMS, EMS, MMS
Connectivity	Mini-USB, Bluetooth
Dimensions	86.3x47x24.4 mm
Weight	96 g
Display	Internal: 320x240 (262k TFT) Extenal: 96x80 (65K CSTN)
Operating System	Motorola
Chipset	i250S1

Key Features

- 2 Cameras: a internal VGA camera with 4x zoom (for 2-way video calling) and a external 2 megapixel camera with 8x zoom.
- Progressive downloading to view media files on demand
- Support of AAC+, MPEG4, WMA, MP3 and Real Video/Audio files
- Enhanced graphics processor for high performance 3D graphics and J2ME™.
- Up to 512M of removable optional Micro SD/TransFlash™ memory card
- Bluetooth supporting wireless stereo sound

J2ME™ Information

CLDC v1.1 and MIDP v2.0 compliant	
Heap size	800 Kb
Maximum record store size	64 K
MIDlet storage available	64MB
Interface connections	HTTP, Socket, UDP, Serial port
Maximum number of sockets	4
Supported image formats	.PNG, .JPEG
Double buffering	Supported
Encoding schemes	ISO8859_1, ISO10646
Input methods	Multitap, iTAP
Additional API's	JSR 75, JSR 82, JSR 84, JSR 120, JSR 135, JSR 139, JSR 184, JSR 185, JSR 205
Audio	MP3, AAC, AAC+, XMF, RA v9 MIDI, WAV

Related Information

Motorola Developer Information:
Developer Resources at
http://www.motocoder.com

Tools:
J2ME™ SDK version v4.0
Motorola Messaging Suite v1.1

Documentation:
Creating Media for the Motorola V3x Handset

References:
J2ME™ specifications: http://www.java.sun.com/j2me
MIDP v2.0 specifications:
http://www.java.sun.com/products/midp
CLDC v1.0 specifications:
http://www.java.sun.com/products/cldc
WAP forum: http://www.wap.org
MMS standards: http://www.3GPP.org

Purchase:
Visit the Motocoder Shop at
http://www.motocoder.com/
Accessories: http://www.motorola.com/consumer

Appendix H: Quick Reference