Technical Manual

MOTOROLA

Developer Guide

MOTORAZR V3xx
Java ME Developer Guide

Version 01.00

Table of Contents

Index of Tables

Index of Figures

Index of Code Samples

1 Introduction

1.1 Purpose

This document describes the application program interfaces used to develop Motorola compliant Java™ Platform, Micro Edition (Java ME) applications for the MOTORAZR V3xx handset supporting CLDC 1.1.

For more detailed information see *Section 3.1.1*.

1.2 Audience

This document is intended for general Java ME developers involved in the production of Java ME applications for the MOTORAZR V3xx handset.

1.3 Disclaimer

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

Motorola makes no warranty in 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 application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

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

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore 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, the 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 manufacturing of the product or service.

Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

1.4 References

Reference	Link
Borland	*http://www.borland.com/*
GSM 03.38 standard	*http://www.etsi.org*
GSM 03.40 standard	*http://www.etsi.org*
IBM	*http://www.ibm.com/*
JSR	*http://www.jcp.org*
MOTODEV	*http://developer.motorola.com*
Motorola	*http://www.motorola.com/*
RFC 2068	*http://www.ietf.org/rfc/rfc2068.txt*
RFC 2396	*http://www.ietf.org/rfc/rfc2396.txt*
RFC 822	*http://www.ietf.org/rfc/rfc822.txt*
SAR	*http://www.wapforum.org*
SSL protocol version 3.0	*http://wp.netscape.com/eng/ssl3/ssl-toc.html*
Sun Java™ ME	*http://java.sun.com/javame/*
Sun Microsystems	*http://www.sun.com/*
Sun MIDP 2.0 SDK	*http://java.sun.com/products/midp/*
TLS protocol version 1.0	*http://www.ietf.org/rfc/rfc2246.txt*
X.509	*http://www.ietf.org/rfc/rfc2459.txt*

Table 1 References

1.5 Revision History

Version	Date	Reason
00.01	03-OCT-2006	Initial Draft.
01.00	18-DEC-2006	Document release.

Table 2 Revision History

1.6 Definitions, Abbreviations, Acronyms

Acronym	Description
(U)SAT	Universal SIM Application Toolkit
AID	Application Identifier
AMS	Application Management Software
APDU	Application Protocol Data Unit
API	Application Program Interface
BCC	Bluetooth Control Center
BMP	Windows BitMap Format (image extension '.bmp')
CLDC	Connected Limited Device Configuration
DNS	Domain Name System
GIF	Graphics Interchange Format (image extension '.gif')
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IMEI	International Mobile Equipment Identity
IP	Internet Protocol
IRCOMM	Is a specification from the Infrared Design Association (IRDA) and determines how different devices can talk to each other via infrared.
IrDA	Infrared Data Association
ITU	International Telecommunication Union
JAD	Java™ Application Descriptor
JAM	Java Application Manager
JAR	Java™ Archieve. Used by Java™ ME applications for compression and packaging.
Java™ ME	Java Platform, Micro Edition (Java™ ME, formerly J2ME)
JPG	Joint Photographic Experts Group (image extension '.jpg')
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	K Virtual Machine (Java™ ME runtime environment)
L2CAP	Logical Link Control and Adaptation Protocol
LCDUI	Limited Connected Device User Interface
MIB	Motorola Internet Browser
MIDP	Mobile Information Device Profile

MMA	Multimedia API
MSISDN	Mobile Station Integrated Services Digital Network
MT	Mobile Terminated
OBEX	OBject Exchange
OEM	Original Equipment Manufacturer
OTA	Over The Air
PKI	Public-Key Infrastructure
PNG	Portable Network Graphics (image extension '.png')
RFC	Request for Comments
RFCOMM	Radio Frequency Communication-Bluetooth protocol that provides emulation of RS-232 (serial) connection
RMS	Record Management System
RUIM	Removable User Identity Module
SAR	Segmentation & Reassembly
SAT	SIM Application Toolkit
SATSA	Security and Trust Services API
SDAP	Service Discovery Application Profile
SDDB	Service Discovery Database-Local listing of available Bluetooth services on device
SDK	Software Development Kit
SDP	Service Discovery Protocol
SE	Security Element
SIM	Subscriber Identity Module
SMS	Short Message Service
SMSC	Short Messaging Service Center
SPP	Serial Port Profile
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TTP	Trusted Third Parties
UDP	User Datagram Protocol
UI	User Interface
UICC	Universal Integrated Circuit Card
URI	Unified Resource Identifier
URL	Universal Resource Locator
USB	Universal Serial Bus
USIM	Universal Subscriber Identity Module
VM	Virtual Machine
WAP	Wireless Application Protocol
WMA	Wireless Messaging API
X.509	Internet X.509 Public Key Infrastructure Certificate and CRL Profile

Table 3 Definitions, Abbreviations, Acronyms

1.7 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 - Java ME Introduction: This chapter describes the Java ME platform and the available resources on this Handset.
Chapter 3 - Developing and Packaging Java ME Applications: This chapter describes important features to look for selecting tools and emulation environments. It also describes how to package a Java ME 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-75 - PIM and Fileconnection APIs: This chapter describes the JSR-75 APIs implementation requirements that shall replace the earlier implemented Phonebook and Fileconnection APIs requirements.
Chapter 7 - 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 8 - MIDP 2.0 Security Model: This chapter describes the MIDP 2.0 default security model.
Chapter 9 - JSR-120 - Wireless Messaging API: This chapter describes JSR-120 implementation.
Chapter 10 - JSR-135 - Mobile Media API: This chapter describes image types and supported formats.
Chapter 11 - 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.
Chapter 12 - JSR-177 Java ME Security and Trust Services API: This chapter describes the JSR-177, which defines optional packages for the J2ME platform. The purpose of this JSR is to specify a collection of APIs that provides security and trust services by integrating a Security Element (SE).
Chapter 13 - JSR-184 - Mobile 3D Graphics API: This chapter describes the JSR-184 which defines an API for rendering three-dimensional (3D) graphics.
Chapter 14 - JSR-185 - Java™ Technology for the Wireless Industry: This chapter describes Java Technology for the Wireless Industry (JTWI) functionality.
Chapter 15 - JSR-205 - WMA 2.0: This chapter describes the functionality that shall be implemented for the WMA.

Chapter 16 - Java ME™ Access to certificates on SIM and phone memory: This chapter presents the marketing requirements specification to access digital certificates on 'SIM or phone memory' by a Java Virtual Machine (JVM)
Chapter 17 - Prevent Downloading of Large Java MIDlets: This feature makes flexible way of preventing the large JAR files OTA download.
Chapter 18 - Download Midlet through PC: This chapter describes download MIDlets through a PC.
Chapter 19 - Downloading MIDlet through Browser: This chapter describes the performing any downloads on the handset.
Chapter 20 - Record Management System: This chapter describes the Record Management System API.
Chapter 21 - Gaming API/Multiple Key Press: This chapter describes the Gaming API.
Chapter 22 - Network APIs: This chapter describes the Java Networking API and network access.
Chapter 23 - Platform Request API: This chapter describes the platform request APIs.
Chapter 24 - JAD Attributes: This chapter describes what attributes are supported.
Chapter 25 - LCDUI: This chapter describes the Limited Connected Device User Interface API.
Chapter 26 - iTAP: This chapter describes iTAP support.
Chapter 27 - Java.lang Implementation: This chapter describes the java.lang implementation.
Chapter 28 - CommConnection Interface: This chapter describes the CommConnection API.
Chapter 29 - Motorola Get URL from Flex API: This chapter describes the way to access URL stored in FLEX by a java application.
Chapter 30 - Motorola Secondary Display API: This chapter details the capability for J2ME applications to render content to Motorola devices that feature a secondary display.
Appendix A - Key Mapping: This appendix describes the key mapping of the MOTORAZR V3xx handset, including the key name, key code, and game action of all Motorola keys
Appendix B - Memory Management Calculation: This chapter describes the memory management calculations.
Appendix C - FAQ: This appendix provides a link to the dynamic online FAQ.
Appendix D - HTTP Range: This appendix provides a graphic description of HTTP Range.
Appendix F - Spec Sheet: This appendix provides the spec sheet for the MOTORAZR V3xx handset.
Appendix H - Quick Reference: This appendix shows the document quick reference.

2 Java ME Introduction

The MOTORAZR V3xx handset includes the Java™ Platform, Micro Edition, also known as the Java ME platform. The Java ME 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 Java ME platform on devices like the MOTORAZR V3xx handset, 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 Java ME environment and the tools that can be used to develop applications for the MOTORAZR V3xx handset.

2.1 The Java™ Platform, Micro Edition (Java™ ME)

The Java ME 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 Java ME architecture, see Figure 1 , 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.

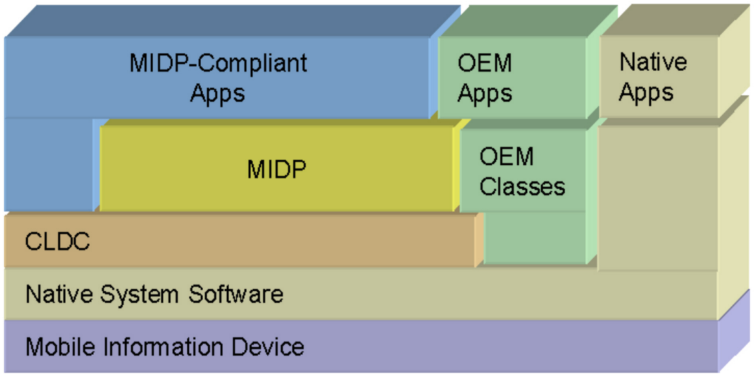


Figure 1 Java™ ME Architecture

The Configuration Layer used in the MOTORAZR V3xx handset 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 Java™ ME, see the Sun Java™ ME documentation (*http://java.sun.com/javame/*).

2.2 The Motorola Java™ ME 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 MOTORAZR V3xx handset contains OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the MOTORAZR V3xx handset can run any application written in standard MIDP, it can also run applications that take advantage of the unique functionality provided by these APIs. These OEM APIs are described in this guide

2.3 Resources and APIs Available

MIDP 2.0 will provide support to the following functional areas on the MOTORAZR V3xx handset:

- 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, .BMP)

Additional Functionality

- JSR-118
- JSR-120
- JSR-135
- JSR-139
- JSR-177
- JSR-184
- JSR-185
- JSR-205
- JSR-75
- JSR-82
- Motorola Get URL from Flex API
- Motorola Secundary Display API

3 Developing and Packaging Java ME Applications

3.1 Guide to Development in Java™ ME

Introduction to Development

This appendix assumes the reader has previous experience in Java ME 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 Java ME handsets.

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

- *http://developer.motorola.com*
- *http://www.java.sun.com/javame*
- *http://www.corej2me.com*
- *http://www.javaworld.com*

As an introduction, brief details of Java ME 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 (JSR - 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.
- For CLDC 1.1; JSR-139 should be reviewed.

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 the 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, arrays of tools are available to assist the development cycle. These range from the command line tools provided with by Software Development Kits (SDK) from Sun to Integrated Development Environments (IDEs) which can be free or purchased. These IDEs come from a range of sources such as Sun, IBM and Borland to name a few.

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 MOTODEV website.

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

3.1.1 Recognizing the phone core specifications

To determine what implementation of MIDP, CLDC and Heap size is on Motorola handset, review the "Java System" details through the menu on the Motorola handset (located under "Java Settings") as shown in Figure 2 .

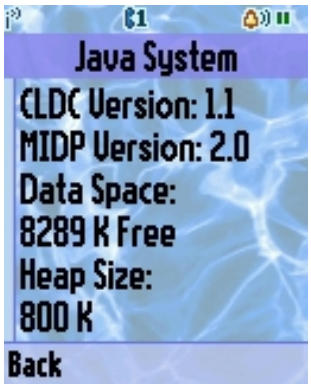


Figure 2 JavaSystem Menu

NOTE: This screenshot is only an example.

4 Downloading Applications

4.1 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 browser. In MIDP , 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 the 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 Opera 8 displaying 'Downloading' followed by "x% completed" for either SAR or HTTP Byte Range type downloads.

A basic Over the Air Server Configuration document can be found in our Technical Articles section at http://developer.motorola.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 MOTODEV called MIDway. The actual version available supports USB cable download. In order to use the tool properly, refer to the FAQ database at *http://developer.motorola.com* portal which contains the following about downloading:

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

In addition to the software a suitable USB-A to Mini-USB cable (Motorola part number SKN6311A can be used).

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 3) on the phone in question and seeing if a "Java app loader" option is available on that menu. If it is not, then contact MOTODEV 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 actual version on the MOTODEV website entitled "Installing Java ME MIDlet using MIDway Tool".

Figure 3 MIDway "Java Tool" menu

The USER-AGENT String

Table 4 describes USER_AGENT strings associated with Motorola devices:

Motorola Device	USER_AGENT STRING
MOTORAZR V3xx	User-Agent: MOT-v3xx/xx.xx.xxR Opera/8 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.

1. WAP Browser Release, Opera 8
2. MIDP version 2.0
3. CLDC version 1.1

4.2 Error Logs

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 sym-	906 Invalid descriptor

		bols) MIDlet-JAR_Size	
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 MOTORAZR V3xx handset. This chapter will discuss the following:

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

5.1 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 confirmation will be obtained before the installation begins. The information presented is obtained from the JAR manifest instead of the JAD.

5.2 MIDlet Upgrade

Rules from the JSR-118 (MIDP 2.0) 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 data is saved if the new MIDlet-version is the same or newer, and if the new MIDlet-data-space requirements are 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 the 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.

5.3 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 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 the application currently in the handset. An older version cannot be imnstalled.
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-75 - PIM and Fileconnection APIs

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

NOTE: Java ME ™ PIM and Fileconnection APIs should be implemented on Java ME ™ platforms supporting CLDC 1.1 and MIDP 2.0 or higher.

6.1 PIM API

The primary goal of the PIM APIs is to provide access to Personal Information Management (PIM) data on Java ME ™ enabled devices. PIM data is defined as information included in the address book, calendar application, and to do list applications.

This chapter still details requirements for implementing Personal Information Management (PIM) and Fileconnection APIs specified in JSR-75 for Java ME ™ enabled mobile devices.

6.1.1 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.EventList.read* - should enable reading the event information available on the device (hereinafter just "event read permission")
- *javax.microedition.pim.EventList.write* - should enable updating the event information available on the device (hereinafter just "event 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 7:

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
User Data Read Capability	Always Ask, Ask Once Per App, Never Ask, No Access	Always Ask	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 7 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.

6.1.2 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_LOCALITY, ADDR_REGION, ADDR_POSTALCODE, ADDR_COUNTRY, ADDR_EXTRA, PHOTO_URL.

- At least the following attributes should 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.

6.1.3 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

6.1.4 To Do List

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

6.2 Fileconnection 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.

6.2.1 Requirements

Fileconnection 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.
- Fileconnection 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 Table 8:

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
User Data Read Capability	Always Ask, Ask Once Per App, Never Ask, No Access	Always Ask	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 8 Groups and permissions for

The Fileconnection 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).
  - Ex: *open("file:///phone/...");*
- "Card" (when the file is stored on a MMC, SD, T-Flash or other card-related media).
  - Ex: *open("file:///SD/...");*

7 JSR-82 - Bluetooth API

7.1 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 required to establish general Bluetooth connections. The `javax.obex` package is required 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.

7.2 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.

7.2.1 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)

7.2.2 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 authorization of connection requests
- The base security settings of the local device, including the security modes defined in the Bluetooth specification

7.2.3 Device Property Table

Table 9 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 is product dependent. The maximum value is 64 Kb.
bluetooth.connected.devices.max	The maximum number of connected devices supported (includes 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.scan	Is inquiry scanning allowed during connection? Valid values are either "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 9 Motorola Bluetooth device properties

7.2.4 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.

7.2.5 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)

7.2.6 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 any 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.

7.2.7 Security Policy

Applications MUST be granted permission to perform any requested operation using this API. Table 10 assigns individual permission to the function groups:

Bluetooth API JSR-82		
Permission	**Protocol**	**Function**
javax.bluetooth	Bluetooth	Data Networking
javax.microedition.io. Connector.bluetooth .client	Bluetooth	Data Networking
javax.microedition.io. Connector.bluetooth. server	Bluetooth	Data Networking
javax.microedition.io. Connector.obex.client	Bluetooth	Data Networking
javax.microedition.io. Connector.obex.server	Bluetooth	Data Networking

Table 10 Security Policy

7.2.8 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 call:

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

7.2.9 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.

7.2.10 Pressing of End Key

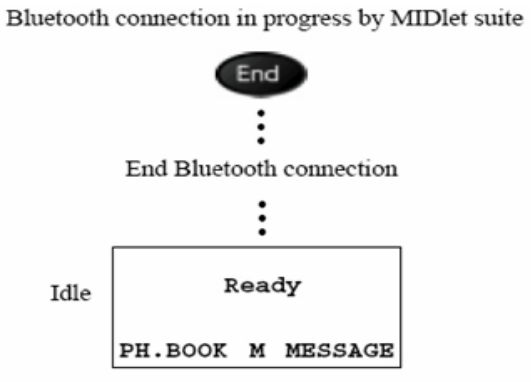


Figure 4 Pressing of End Key

Rules:

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

7.2.11 Hardware Requirements

Requires Java ME and Bluetooth wireless technology for the javax.bluetooth support.

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

7.2.12 Interoperability Requirements

SDK Developer/Style Guide Requirements:

The following table lists the suggested types of screens and text used for user feed-back.

Examples of each screen type are provided below.

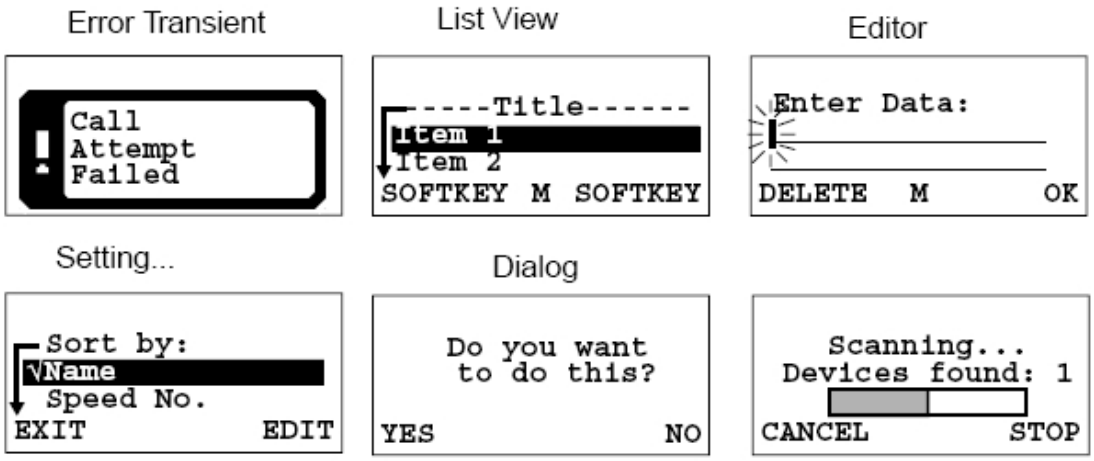


Figure 5 Examples Screens

Event	Screen	Text	LSK	RSK	Title
BCC	List	N/A	BACK	SELECT	"Bluetooth Link"
Discoverable	Dialog	"Discover-able..." <Timer Countdown>	CANCEL	RETURN	N/A
Bond Re-quest	Dialog	"Bond with <device>?"	YES	NO	N/A
Invalid PIN	Transient	"Invalid PIN"	N/A	N/A	N/A
Service Dis-covery	Dialog	"Scanning..." "Devices found: <#>" <Progress Meter>	CANCEL	STOP	N/A
Name Dis-covery	Dialog	"Retriev-ing..." "Device Names: x / #"	CANCEL	STOP	N/A
Device His-tory	List	N/A	BACK	LINK	"Bluetooth Link"
No Devices Found	Transient	"No Devices Found"	N/A	N/A	N/A
New Devices	List	N/A	BACK	LINK	"Scan Res-ults"
PIN Entry	Editor	N/A	DELETE	OK	"Enter PIN"

Table 11 Interoperability Requirements

8 MIDP 2.0 Security Model

The following sections describe the MIDP 2.0 Default Security Model for the MOTORAZR V3xx 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 MOTODEV website (*http://developer.motorola.com*).

Refer to Table 12 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
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 SHA-1 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	Supported

javax.microedition.pki package	
Will preload two self authorizing Certificates	Supported
All constructors, methods, and inherited methods for the MIDletStateChangeException class in the javax.microedition.midlet package	Supported
All constructors and inherited methods for the MIDletStateChangeException class in the javax.microedition.midlet package	Supported

Table 12 MIDP 2.0 Feature/Class

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

8.1 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 one or more 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.

8.2 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

8.3 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 MOTORAZR V3xx 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 MOTORAZR V3xx, 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.

8.4 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 permission 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 permission that requires 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.

8.5 User Permission Interaction Mode

User permission for the MOTORAZR V3xx 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 the 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. (**Ask Once Per App**)
- 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 runtime options.

8.6 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.

8.7 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.

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.

Table 13 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 13 Trusted 3rd Party Domain

The radio button messages will appear as follows and mapped to the permission types as shown in Table 14:

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 14 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.

8.8 Security Policy for Protection Domains

Table 15 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 Acess	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	**Always Ask**, Ask Once Per App, No Acess	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	**Ask once Per App**, Always Ask, Never Ask, No Access	**Always Ask**	Full Access	Full Access
User Data Read Capability	**Ask once Per App**, Always Ask, Never Ask, No Access	**Always Ask**	Full Access	Full Access
User Data	**Ask once**	**No Access**	Full Access	Full Access

Write Capability	**Per App**, Always Ask, Never Ask, No Access			
Multimedia Recording	**Ask once Per App**, Always Ask, Never Ask, No Access	**No Access**	Full Access	Full Access

Table 15 Security Policy for Protection Domains

Table 16 shows individual permissions assigned to the function groups shown in the above table .

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 16 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.

8.9 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.

8.10 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).

8.11 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 MOTORAZR V3xx handset.

8.12 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).

8.13 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 must 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.

8.14 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.

8.15 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.

8.16 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 for 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.

8.17 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.

8.18 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 at 1 and is incremented by 1 until there is no attribute with this name. The value of each attribute is a base64 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> is 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.

Table 17 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 17 Actions performed of signer certificate verification

8.19 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 Table 17
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).

Table 18 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 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 18 Summary of MIDlet suite verification

8.20 Carrier Specific Security Model

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

9 JSR-120 - Wireless Messaging API

9.1 Wireless Messaging API (WMA)

Motorola has implemented certain features that are defined in the Wireless Messaging API (WMA) 1.0. 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 Java ME applications using SMS to communicate game moves running on the handset. 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

9.2 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 1.1. 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");
```

9.3 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 Java ME 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 sections 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.

9.4 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 they will be removed, from the inbox by way of a FIFO stack.

9.5 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.

9.6 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 should not send messages that will take up more than three SMS protocol messages unless the recipient's device is known to support more.

9.7 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 Java ME 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

Table 19 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 19 List of Messaging features/classes

Code Sample 1 shows implementation of the JSR-120 Wireless Messaging API:

Creation of client connection, creation of binary message, setting of payload for binary message and 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. BIN-
ARY_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());
```

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 WMA

10 JSR-135 - Mobile Media API

10.1 JSR-135

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

- Tone Sequence
- Sampled Audio
- MIDI
- Interactive MIDI

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

Code Sample 2 shows the 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 MMA

10.2 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

10.3 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.

10.4 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.

10.5 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 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.

10.6 Supported Multimedia File Types

The following section lists media file types (with corresponding CODECs) that are supported in products that are JSR-135 compliant in addition to JSR-135 Mobile API Phase I. The common guideline being all codecs and file types supported by native side are accessible through the JSR-135 implementation.

10.6.1 Audio Media

File Type	Codec
WAV	PCM
WAV	ADPCM
SP MIDI	General MIDI
MIDI Type 0	General MIDI
MIDI Type 1	General MIDI
iMelody	IMelody
CTG	CTG
MP3	MPEG-1 layer III
AMR	AMR
BAS	General MIDI

Table 20 Audio Media

10.6.2 Image Media

File Type	Functionality
JPEG	Playback/Capture
Progressive JPEG	Playback
PNG	Playback
BMP	Playback
WBMP	Playback
GIF 87a, 89a	Playback

Table 21 Image Media

10.6.3 Video Media

File Type	Functionality
H.263	Playback
MPEG4	Playback
Real Video G2	Playback
Real Video 8	Playback
Real Video 9	Playback

Table 22 Video Media

10.7 Media Locators

The classes Manager, DataSource and RecordControl interface accepts media locators. In addition to normal playback locators specified by JSR-135, the following special locators need to be supported:

10.7.1 RTSP locator

RTSP Locators must be supported for streaming media on devices supporting real time streaming using RTSP. This support must be available for audio and video streaming through Manager (for playback media stream).

NOTE: Refer to JSR-135 API for RTSP locator syntax.

10.7.2 HTTP Locator

HTTP Locators must be supported for playing back media over network connections. This support should be available through Manager implementation.

e.g.: Manager.createPlayer("http://webserver/tune.mid")

10.7.3 File Locator

File locators must be supported for playback and capture of media. This is specific to Motorola Java ME 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")

10.7.4 Capture Locator

Capture Locator should be supported for audio and video devices. A new device "camera" must 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")

NOTE: For mandatory capture formats, refer to section 0.0.4. Refer to JSR-135 API for capture locator syntax.

10.8 Security

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

10.8.1 Policy

Table 23 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 Acess	Full Access	Full Access

Table 23 Security policy

10.8.2 Permissions

Table 24 lists individual permissions within Multimedia Record function group.

Permission	Protocol	Function Group
javax.microedition.	RecordCon-	MultimediaRecord

media.control. RecordControl.re	trol.startRecord()	

Table 24 Permissions within Multimedia Record

NOTE: The Audio/Media formats may differ or may not be avaliable, depending on the Carrier or region.

11 JSR-139 - CLDC 1.1

11.1 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 Java SE compliant
- Thread objects to be compliant with Java SE.

The support of thread objects to be compliant with Java SE requires the addition of Thread.getName and a few new constructors. 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 equ-alIgnoreCase	Compares this string to another String, ig-

		(String another-String)	noring case considera-tions
		String intern ()	Returns a canonical representation for the string object
		Static String valueOf (float f)	Returns the string rep-resentation of the float argument
		Static String valueOf (double d)	Returns the string rep-resentation 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
Calendar and Time Classes	Java.util.Calendar	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, ex-pressed 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 ob-ject to a String of the form: Dow mon dd hh:mm:ss zzz yyyy
Exception and Error Classes	Java.lang.NoClassD efFoundError		New Class: Refer to CLDC Spec for more details
Weak References	Java.lang.ref.Refere nce		New Class: Refer to CLDC Spec for more

			details
	Java.lang.ref.WeakReference		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 25 Additional classes, fields, and methods supported for CLDC 1.1 compliance

12 JSR-177 Java ME Security and Trust Services API

12.1 Feature Description

This chapter describes the JSR-177 which defines optional packages for the Java ME platform. The purpose of this JSR is to specify a collection of APIs that provides security and trust services by integrating a Security Element (SE). An SE, provides the following:

- Secure storage to protect sensitive data, such as the user's private keys, public key (root) certificates, service credentials, personal information, etc.
- Cryptographic operations to support payment protocols, data integrity, and data confidentiality.
- A secure execution environment to deploy custom security features. MIDlets would rely on these features to handle many value-added services, such as user identification and authentication, banking, payment, loyalty applications, and so on.

Smart cards (SIM/USIM/UICC/RUIM) are commonly used to implement an SE. For example, on GSM networks, the network operator enters the network authentication data on the SIM, as well as the subscriber's personal information, such as the address book. When the subscriber inserts the SIM into a mobile handset, the handset is enabled to work on the operator's network.

In addition to a SIM card-based implementation, an SE can also be implemented in the handset itself. Such implementation may utilize, for example, embedded chips or special security features of the hardware.

Alternatively, an SE may be entirely implemented in software. This specification does not exclude any of the possible implementations of an SE even though some of the packages are optimized for smart card implementation.

SEs can have diverse software and hardware characteristics, but this specification considers the API functions based on the following criteria:

- Size requirements for resource-constrained consumer devices
- SE scope
- Flexibility and extensibility of the API

Based on these criteria, this version of the JSR 177 defines an API to provide the following capability to the Java ME platforms:

- Smart Card Communication - Two smart card access methods are defined in this specification based on the APDU protocol and the Java Card RMI protocol. These access methods allow a Java ME application to communicate with a smart card to enhance the security services deployed on it.

A MIDlet must be granted permission to use the privileged API in the SATSA-APDU optional package. Permissions are checked by the platform prior to the invocation of the protected methods in the API. Based on the security framework implemented by the underlying platform, an implementation of a SATSA optional package must support either the MIDP 2.0 permissions applicable to that optional package or the functional equivalent Java SE style permission classes. Based on the access control policy defined in a smart card, the device determines whether the MIDlet is allowed to access any function of the smart card, using the APDUConnection.

The smart card communication API is based on the Generic Connection Framework (GCF), which is defined in the CLDC 1.1 specification.

12.2 Assumptions/Dependencies

The system must be compliant to the following standards:

- CLDC 1.1
- MIDP 2.0

12.3 New Implementation

This article will only reference the APDU package of JSR 177, SATSA-APDU since it is the only one implemented on Motorola JSR 177 compliant devices.

The smart card communication API is based on the Generic Connection Framework in the `javax.microedition.io` package. The APDUConnection interface is used to communicate with ISO 7816-4 compliant smart cards.

12.3.1 javax.microedition.apdu Optional Package

The SATSA-APDU optional package supports the following functions:

- A MIDlet can create an APDUConnection to communicate with a smart card application identified by an AID.
- An APDUConnection supports exchange of APDU commands encoded in the format, which conforms to ISO 7816-4.
- Each APDUConnection has a logical channel reserved exclusively for it.
- Logical channel management is handled by the API implementation, which requests the smart card to allocate an unused logical channel.
- More than one APDUConnection can be created to communicate simultaneously with smart card applications on one (via logical channels) or multiple smart cards.
- An APDUConnection can be created to communicate with (U)SAT applications on channel 0 of a smart card. The APDUConnection has limited capabilities when communicating with a (U)SAT application.
- Only ENVELOPE APDUs may be sent by a MIDlet to trigger a (U)SAT application.
- Proactive sessions and commands are not supported by the APDUConnection. It is the MIDlet's responsibility to not send an envelope to the (U)SAT application that would result in a proactive session being initiated via the APDUConnection interface.

The optional package SATSA-APDU includes two components to support communication with ISO 7816-4 compliant smart cards using the APDU protocol:

- A subset of the `java.lang` package. It supports the exception class `UnsupportedOperationException`, which is not included in the CLDC API or the MIDP API.
- The `javax.microedition.apdu` package. It includes the interface APDUConnection to support APDU exchanges.

APDUConnection Interface

This interface extends the `javax.microedition.io.Connection` interface. It defines the APDU connection. MIDlets can use this connection to communicate with applications on a smart card using APDU protocol. ISO 7816-4 defines the APDU protocol as an application-level protocol between a smart card and an application on the device. There are two types of APDU messages:

- command APDUs - sent to the smart card by a MIDlet
- response APDUs - messages received from the smart card

All APDUs is generated according to the ISO 7816-4 specification.

Opening an APDU Connection

An APDU connection is established between a MIDlet and an application on a smart card. It uses one dedicated logical channel for that connection. The implementation supports multiple APDU connections between the handset and a smart card. Additionally it supports multiple APDU connections between a MIDlet and an application on a smart card (if multiple connections are supported by the application on a smart card).

A MIDlet uses method `javax.microedition.io.Connector.open()` to open an APDU connection. An APDU connection is created by passing a generic connection URI string with the smart card AID and, optionally, the slot in which the smart card is inserted, to the `javax.microedition.io.Connector.open()` method. When the method `javax.microedition.io.Connector.open()` is called by a MIDlet, the implementation requests a new logical channel from the smart card, for the APDU connection, unless channel 0 is used. When method `javax.microedition.io.Connector.open()` is called by a MIDlet, and the logical

channel is acquired for the APDU connection, the implementation establishes an APDU connection with the specified application and returns the resulting connection object to the MIDlet.

The implementation uses method `system.getProperty()` (with key `javax.microedition.smartcardslots`) to obtain the name of the smart card slots to be used in opening an APDU connection.

The value returned is a comma-separated list of the smart card slots which can be used in the `javax.microedition.io.Connector.open()` string to identify the specific smart card slot. The logical slot names include the slot number and a descriptor indicating the type of the slot. For cold-swappable slots the letter 'C' is appended to the slot number. For hot-swappable slots the letter 'H' is appended to the slot number. The slot descriptors (the letter 'C' and 'H' appended to the slot number) cannot be passed as part of the URI to open a connection to the smart card application. The Java ME application MUST remove the descriptor from the logical slot name and only use the slot number in the URI to identify the specific smart card slot. The (U)SIM card will by default always be in slot (name) 0.

If the opening of the APDU connection fails, the implementation releases the logical channel established between the handset and the smart card for the APDU connection.

The URI used in method `javax.microedition.io.Connector.open()` conforms to the following BNF syntax:

URI Format Description	BNF Syntax
<APDU_connection_string>	::= "apdu:"<targetAddress>
<targetAddress>	::= [slot];target
<slot>	::= smart card slot number. (optional. Hexadecimal number identifying the smart card slot. Default slot assumed if left empty)
<target>	::= "target="<AID>|"SAT"
<AID>	::=< 5 - 16 bytes > An AID uniquely identifies a smart card application. It is represented by 5 to 16 hexadecimal bytes where each

	byte value is separated by a "."

Table 26 javax.microedition.io.Connector.open() BNF syntax

APDU Connection Establishment Errors

If a logical channel is not available for the APDU connection, the implementation throws the `IOException` to the MIDlet.

If a MIDlet calls method `javax.microedition.io.Connector.open()` with a nonexistent card slot number, the implementation throws `ConnectionNotFoundException` exception.

If a MIDlet calls method `javax.microedition.io.Connector.open()` with a card slot number which does not have a card in it, the implementation throws `ConnectionNotFoundException` exception.

If a MIDlet calls method `javax.microedition.io.Connector.open()` with a nonexistent AID, the implementation throws `ConnectionNotFoundException` exception.

If a MIDlet calls method `javax.microedition.io.Connector.open()` and the target application on the card refuses connection request, the implementation throws `ConnectionNotFoundException` exception.

If a MIDlet calls method `javax.microedition.io.Connector.open()` but the MIDlet is not allowed to access the targeted application on the card, the implementation throws `SecurityException` exception.

Using an APDU Connection

Interface APDUConnection implements method `exchangeAPDU()`.

The implementation interprets the CLA byte of the commandAPDU parameter passed in the `javax.microedition.apdu.APDUConnection.exchangeAPDU()` method as the logical channel number.

Once an APDU connection is created, a MIDlet can use the `javax.microedition.apdu.APDUConnection.exchangeAPDU()` method to send com-

mand APDUs and receive response APDUs to and from the smart card.

The implementation services only one logical channel (one APDU connection) at a time, i.e. current APDU procedure on channel A must complete before a new APDU procedure can begin on channel B.

There may be several APDU connections open at the same time using different logical channels with the same smart card. However, since APDU protocol is synchronous, there can be no interleaving of command and their response APDUs across logical channels. Between the receipt of the command APDU and the sending of the response APDU to that command, only one logical channel is active.

Interface APDUConnection implements method `javax.microedition.apdu.APDUConnection.changePin()`.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.changePin()`, the implementation prompts the user to enter the existing value of the PIN, the new value for the PIN, and to re-enter the new value for the PIN for confirmation.

Interface APDUConnection implements method `javax.microedition.apdu.APDUConnection.disablePin()`.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.disablePin()`, the implementation prompts the user to enter the value of the PIN that is to be disabled.

Interface APDUConnection implements method `enablePin()`.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.enablePin()`, the implementation prompts the user to enter the value of the PIN that is to be enabled.

Interface APDUConnection implements method `enterPin()`.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.enterPin()`, the implementation

prompts the user to enter the current value of the PIN, for verification purposes.

Interface APDUConnection implements method `unblockPin()`.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.unblockPin()`, the implementation prompts the user to enter the value of the unblocking PIN (PIN used to unblock other PINs), the new value for the currently blocked PIN, and to re-enter the new value for the currently blocked PIN.

The implementation communicates the entered PIN value(s) to the card over the APDU connection, and returns the smart cardstatus, or NULL if the user canceled the operation.

Interface APDUConnection implements method `getATR()`.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.getATR()`, the implementation returns the retrieved value of ATR from the smart card.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.getATR()`, the implementation returns NULL if the smart card is not present.

When a MIDlet calls method `javax.microedition.apdu.APDUConnection.getATR()`, the implementation returns NULL if the APDU connection was already closed.

If response '61 XX' is received from the smart card, the implementation sends GET RESPONSE to the card to get the response data before any other command is sent.

If response '6C XX' is received from the smart card, the implementation resends the command after setting "Le" equal to XX received from the smart card before any other command is sent.

Errors While Using APDU Connection

If a MIDlet attempts to exchange APDUs and the connection was closed before the method was called or because of communication problems, the implementation

throws `IOException` exception.

If a MIDlet attempts to exchange APDUs using the connection object created before the card was removed and then reinserted the implementation throws `InterruptedIOException` exception.

If a MIDlet attempts to exchange APDUs on an APDU connection that is closed during the communication session, the implementation throws `InterruptedIOException` exception.

If a MIDlet calls method `exchangeAPDU()` and the commandAPDU parameter is NULL, the implementation throws `java.lang.IllegalArgumentException` exception.

If a MIDlet calls method `exchangeAPDU()` and the commandAPDU contains a card application selection APDU, the implementation throws `java.lang.IllegalArgumentException` exception.

If a MIDlet calls method `exchangeAPDU()` and the commandAPDU parameter contains a MANAGE CHANNEL command APDU, the implementation throws `java.lang.IllegalArgumentException` exception.

If a MIDlet calls method `exchangeAPDU()` and the channel associated with the connection object is non-zero and the CLA byte has a value other than 0x0X, 0x8X, 0x9X or 0xAX, the implementation throws `java.lang.IllegalArgumentException` exception.

If a MIDlet calls method `exchangeAPDU()` and the commandAPDU parameter contains a malformed APDU, the implementation throws `java.lang.IllegalArgumentException` exception.

Closing an APDU Connection

If a MIDlet calls method `javax.microedition.io.Connection.close()`, the implementation closes the APDU connection.

When method `javax.microedition.io.Connection.close()` is called by a MIDlet, the implementation releases the logical channel used by the APDU connection.

If the connection that was closed was using channel 0, the implementation updates

it's availability status.

Logical channels other than the basic channel may be closed when the connection is closed. Basic channel or channel 0 has to remain open, and cannot be closed.

Error Cases When Closing APDU Connection

If a MIDlet calls method `javax.microedition.io.Connection.close()` on a connection that is executing in another thread, any pending I/O method throws `InterruptedIOException`.

The methods of APDUConnection are not synchronized. The only method that can be called safely in another thread is `javax.microedition.io.Connection.close()`.

If a MIDlet terminates without calling `javax.microedition.io.Connection.close()` on the open connection, the implementation performs the close operation automatically in order to recover resources such as the logical channel.

Support for (U)SIM Application Toolkit ((U)SAT)

When a MIDlet calls method `javax.microedition.io.Connector.open()` to open a connection to a (U)SAT application, the implementation acquires channel 0.

When a MIDlet calls method `javax.microedition.io.Connector.open()` to open a connection to a (U)SAT application, the implementation addresses (U)SIM in slot 0.

The implementation ensures that channel 0 is always available for the new APDU connection to the (U)SAT.

When a MIDlet calls method `javax.microedition.io.Connector.open()` to open a connection to a (U)SAT application, the URI string is formatted as "apdu:<slot>;target=SAT".

A MIDlet in the operator domain has full and exclusive access to the APDU connection.

If the MIDlet is trying to access PIN-related methods while a APDU connection is established with a (U)SAT application, the implementation throws `SecurityException`.

A MIDlet uses ENVELOPE APDUs only, to communicate with the (U)SAT application.

If a MIDlet uses any APDU other than ENVELOPE APDU, the implementation throws `IllegalArgumentException` exception.

In the case of a SIM (GSM) smart card, the implementation substitutes the class byte passed by the MIDlet with value 'A0'.

In the case of a USIM (UMTS) smart card, the implementation substitutes the class byte passed by the MIDlet with value '80'.

When (U)SIM responds with status word "9E XX"or "9F XX", the behavior of APDUConnection is the same as when status word "61 XX" is received from the smart card.

When (U)SIM responds with status word "62 XX" or "63 XX" the implementation sends GET RESPONSE to the card, with Le set to "00" before any other command is sent.

The implementation ensures that between sending the ENVELOPE APDU, receiving status word "62 XX" or "63 XX", and sending GET RESPONSE APDU with Le set to "00", there will not be any other APDU exchange on any logical channel with the smart card.

The implementation throws `IOException` exception if (U)SIM responds with status word "93 00" (SIM Application Toolkit is busy) when the (U)SIM is performing another proactive session.

12.3.2 java.lang Package (Exception Classes)

The package provides classes that are fundamental to the design of the Java programming language.

Class `UnsupportedOperationException` is implemented, as specified in java.lang package.

12.3.3 Recommended Security Element Access Control

Access control governs the establishment of an APDU connection and communication using the APDU connection between terminal objects and on SE objects.

The access control model is designed to achieve the following security objectives:

- Protect an SE from malicious MIDlets
- Support the SE to specify a fine-grained access control policy within the limitations of the platform
- Allow a MIDlet to select an SE object (for example, a smart card application) for temporary exclusive usage
- Safeguard PINs from improper usage by the MIDlets

The system will use two mechanisms to implement access control: the Domain Mechanism and the Static Mechanism, for all SEs on the handset.

In the Domain Mechanism, an SE defines a private domain by providing the domain root object (trusted certificate or public key). In the Domain Mechanism, the SE accepts only access from MIDlets that reside in such a domain (i.e., the application is signed with a certificate that chains back to the trusted certificate provided by the SE).

In the Static Mechanism, an ACF is published by an SE. The ACF contains access control for individual methods, and applications on the SE. ACFs are stored in the SE. The terminal platform is responsible for processing these files.

The implementation reads the certificate from the SE.

The implementation reads Access Control Files from the SE.

Each SE has one ACIF associated with it. Each ACIF contains a list of ACFs (an ACIE), one for each application on the SE. Each ACF may contain a list of zero or more ACEs (an ACL).

When a MIDlet calls a method, the implementation evaluates if the MIDlet has appropriate permissions to access it, by first applying the Domain Mechanism.

The implementation applies the Domain Mechanism according to MIDP 2.0 and se-

curity policy requested by the operator.

The implementation applies the following algorithm when evaluating a request for access:

Request for Access Algorithm

```
Use Domain Mechanism (DM) to evaluate access to the method

if access is forbidden according to DM
       Access Denied
else /* Access Granted according to DM, check ACP now */
       if PKCS#15 is present
              if ACIF link in DODF is present, then
                     if ACIF is found, then
                            loop for ACIE
                                   if either ACIE contains AID for the target app or ACIE related to the entire SE
                                          get corresponding ACF to evaluate access permissions
                                          if ACL is present
                                          get ACE and evaluate access to the method
                                          else
                                          Access Granted /* ACL is missing */
                                          endif
                                   endif
                            endloop
                            Access Denied /* either no ACIE element is found or all found ACLs are empty */
                     endif /* for ACIF */
                     Access Denied /* ACP link exists, but no applicable ACIF is found */
              else /* No ACP */
                     Access Granted
              endif
       else /* No PKCS#15 */
              Access Granted
       endif
endif


If any errors in ACP or PKCS#15 (U)SIM structure is found during the evaluation algorithm
execution access is denied.
```

Code Sample 3 Request for Access Algorithm

The algorithm can be implemented in many different ways. The most efficient one should be chosen. Implementers are not limited to using if-else statements.

If the MIDlet is trying to access a method protected by the Domain Mechanism, and access to it is denied, the implementation throws `SecurityException`.

If the MIDlet is trying to access a method protected by the Static Mechanism, and access to it is denied, the implementation throws `SecurityException`.

Evaluating Individual Access Control Entry

When evaluating ACE, the MIDlet is granted permission to open an APDU connection with an application in the SE if the ACE principal identifies a domain category (CHOICE domain is used with the OID indicating 'operator', 'manufacturer', or 'trusted third party') and the MIDlet belongs to the same domain.

When evaluating ACE, the MIDlet is granted permission to open an APDU connection with an application in the SE if the ACE principal identifies the domain root (CHOICE rootID is used) and the corresponding PrincipalID matches with the hash of the root certificate in the path used to sign the MIDlet.

When evaluating ACE, the MIDlet is granted permission to open an APDU connection with an application in the SE if the ACE principal identifies an end-entity (CHOICE endEntityID is used) and the corresponding PrincipalID matches with the end-entity certificate used to sign the MIDlet.

When evaluating ACE, the MIDlet is granted permission to send an APDU to an application in the SE if the APDU being sent by the MIDlet is specified by at least one ACE.

When evaluating ACE, the MIDlet is granted permission to send an APDU to an application in the SE if the APDU being sent by the MIDlet is not one of those used for application selection and channel management.

A MIDlet operation is considered to be specified by an ACE if the following condition is satisfied: APDU(MIDlet) AND mask(ACE) = APDU(ACE),

12.3.4 Security Requirements

The `javax.microedition.io.Connector.open` ("apdu:"[<slot>]",target=SAT") method is protected by the `javax.microedition.apdu.sat` permission.

The permission `javax.microedition.apdu.sat` is not assigned to any function group.

The access to methods in `javax.microedition.apdu.sat` package is granted only to MIDlets in the operator's domain.

The `javax.microedition.io.Connector.open` ("apdu:"[<slot>]",target="<AID>) method is protected by the `javax.microedition.apdu.aid` permission.

The permission `javax.microedition.apdu.aid` is assigned to the Smart Card Communication function group.

The default security policy supports the Ask Once Per App, Never Ask, No Access permission interaction modes for the Smart Card Communication function group for the TTP domain. Ask Once Per App interaction mode is the default.

The default security policy supports only No Access permission interaction mode for the Smart Card Communication functional group for the Untrusted domain.

13 JSR-184 - Mobile 3D Graphics API

13.1 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 Java ME device supporting MIDP 2.0 and CLDC 1.1 as a minimum.

13.2 Mobile 3D API

The MOTORAZR V3xx contains full implementation of JSR-184 Mobile 3D API (*http://jcp.org/en/jsr/detail?id=184*). The MOTORAZR V3xx 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 an 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.

13.3 Mobile 3D API File Format Support

The MOTORAZR V3xx supports both M3G and PNG file formats for loading 3D content. The MOTORAZR V3xx 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.

13.4 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.

13.4.1 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 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.

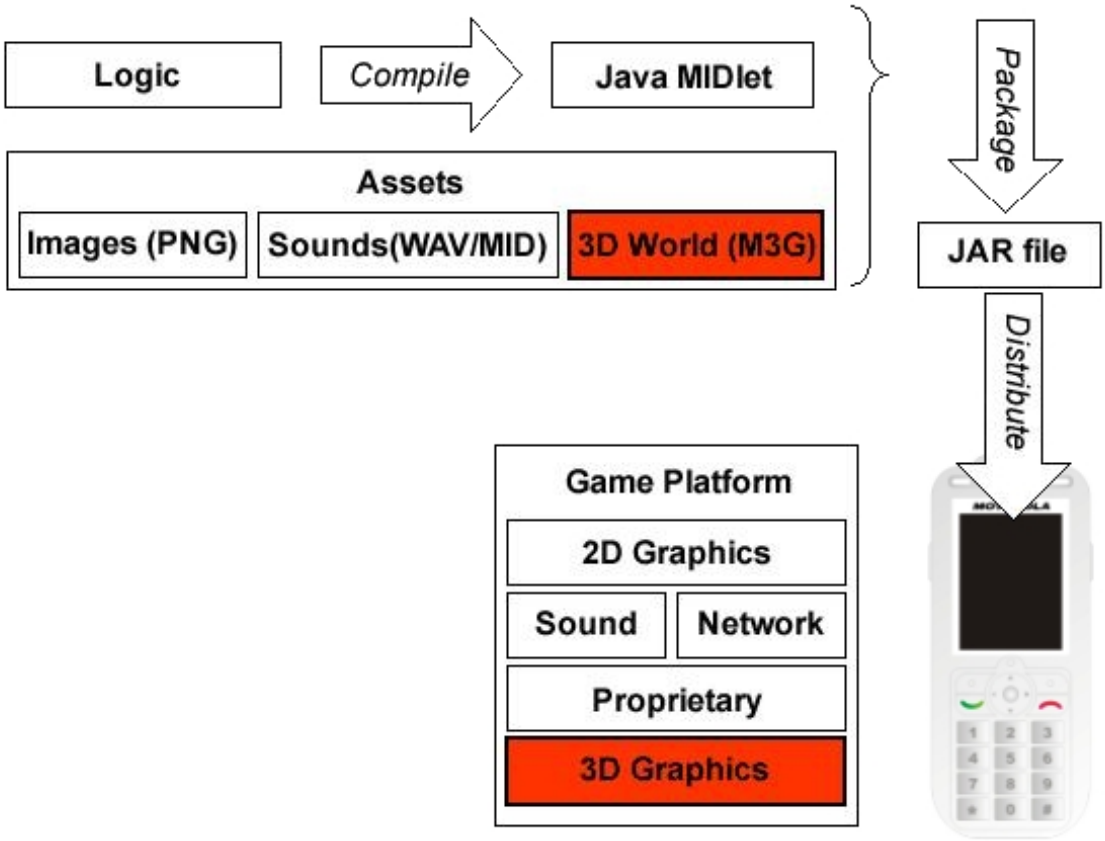


Figure 6 M3G Application Proccess

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.

13.4.2 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.

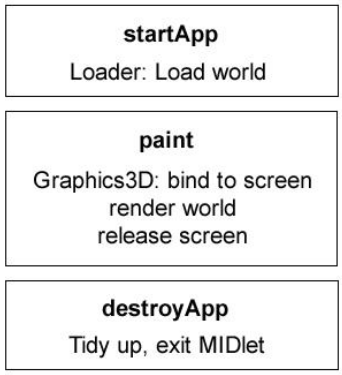


Figure 7 M3G Application Methods

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.

Figure 8 shows the structure of a more typical MIDlet.

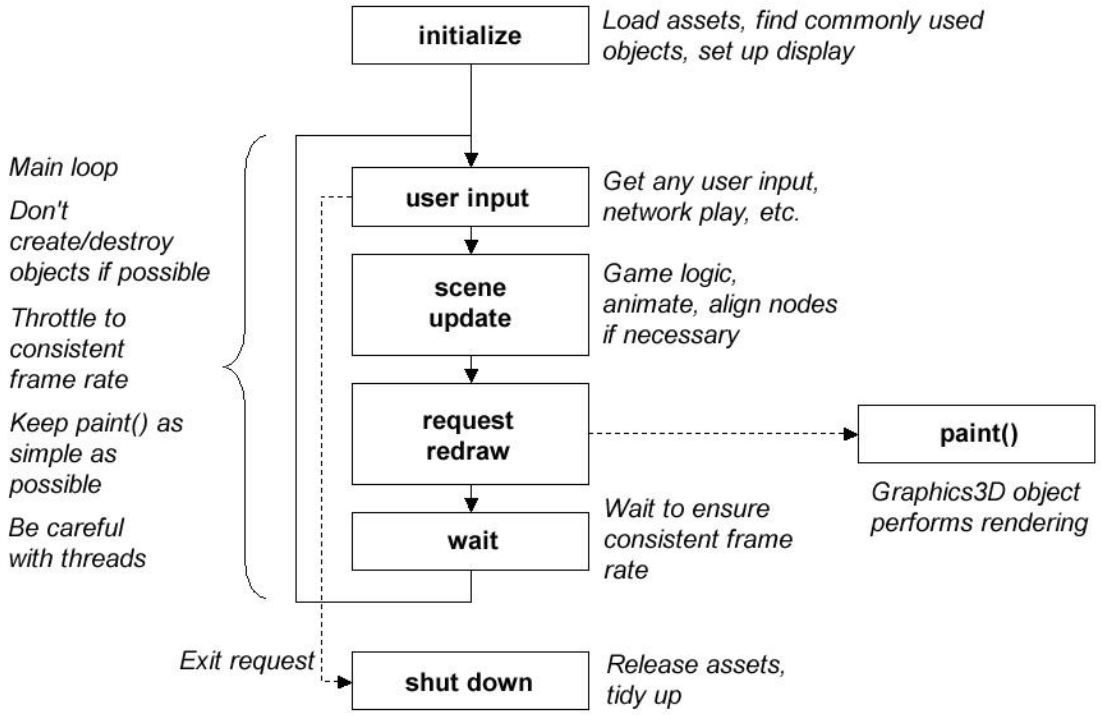


Figure 8 Typical MIDlet Structure

13.4.3 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 4 Initializing the world

13.4.4 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 5 Using the Graphics3D object

The final block makes sure that the target is released and the Graphics3D can be re-used. 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.

13.4.5 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);
```

Code Sample 6 Finding objects by 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]);

	}

}
```

Code Sample 7 Using the Object3D.getReferences().

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.

13.4.6 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 Animation-Track. 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);
```

Code Sample 8

13.4.7 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 SwerveTM Studio or SwerveTM 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/*.

14 JSR-185 - Java™ Technology for the Wireless Industry

Java™ Technology for the Wireless Industry (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.

14.1 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 screen size of 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)

14.2 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 preferred name has not 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 to use of custom time zones which adhere to the following time zone 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 59. 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

14.3 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 JTWI requirements for MIDP 2.0 will support the following points:

- 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

14.4 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

14.5 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.

14.6 MIDP 2.0 Security specific content for JTWI

- The MOTORAZR V3xx follows the security policy outlined in the Security chapter of this developer guide.

15 JSR-205 - WMA 2.0

15.1 Overview

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

15.1.1 Messaging Functionality

This section describes messaging functionality to be implemented by WMA.

15.1.2 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.

15.1.3 MMS Message Addressing

The multipart message addressing model contains different types of addresses:

- global telephone number of recipient user, including telephone number, ipv4, ipv6 addresses
- e-mail address as specified in RFC 822
- 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 the JSR-205 specification.

15.1.4 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 assumed 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.

15.1.5 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 specifying the subject of the message.

Please refer to JSR-205 specification for more details.

15.1.6 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.

15.1.7 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 to the JSR-205 specification for more details.

15.1.8 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 to 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.

15.1.9 MMS Push

The registration for MMS-push mechanism and MMS-push mechanism itself shall be implemented, but applied to MMS messages in addition to what is described in the MIDP 2.0 chapter. This includes push registry and all user dialogues.

When an application that is registered in the Push Registry is deleted, the corresponding PUSH entry shall be deleted and the corresponding application ID shall be made available for future PUSH registrations.

15.2 Requirements for WMA

The WMA shall accept the application-ID allocated by the first application. If other applications try to allocate the same application-ID while it is being used by the first application, an IOException shall be thrown when they attempt to open the MessageConnection. The same rule applies if an application-ID is being used by a system application in the device. In this case, the Java application will not be able to use that application-ID.

MMS-push mechanism shall be implemented as described in the MIDP 2.0 chapter and some specific requirements are defined below in this section.

15.2.1 Initial Setup

The MMS initial setup parameters set by user shall not be accessible by WMA. The initial MMS setup requirements are outside the scope of this document.

The Java Client will use the MMS Setup of the native client to send/receive messages. So the Java client will use the same APN/Web-Sessions/mmsc etc. as the Native Client.

15.2.2 Handling the incoming MMS message

WMA shall be responsible for listening to the inbound connections for incoming MMS messages with registered Application IDs.

The WMA shall launch the MMS application and suspend listening of incoming MMS messages for this Application ID. Then the application is responsible for the handling of inbound connections (open/close) for the MMS messages (receive/send).

Once the Application exits (terminated, or not successful launch, or user denied the MMS application launch) then WMA shall resume the listening of the inbound connections.

The incoming MMS messages shall be stored in a separate FIFO message Inbox that is not visible to the user. The amount of memory allocated for this transparent inbox is product specific.The MMS application Inbox shall not be accessible for native MMS application.

The WMA shall pass the received MMS messages to concrete Java application assosiated with the Application-ID.

Application running/resuming

The Application startup and resume shall be implemented in accordance with requirements outlined in the MIDP 2.0 chapter.

If an MMS application startup was denied by the user then WMA shall remove all buffered unread messages for this MMS application.

Application is running/background

The Application receiving the incoming MMS message will handle this MMS message.

When the MMS message is received by an application, it shall be removed from the Phone/SIM memory where they may have been stored prior to being delivered to the application.

An application is responsible to handle a corresponded (Application ID) received message (store it more persistently if needed). An MMS message may get lost if an application can not save it due to lack of space.

The application will be responsible for the interpretation and representation of the MMS MIME content including the SMIL(presentation) content if any is attached.

In the case of full incoming message buffer, any new message for the application with the same Application-ID shall be discarded. WMA shall not remove the first MMS message in the buffer which was a cause of Push until:

- MMS message is handled by MMS application, or
- MMS application exits.

Application suspending

The Application suspending shall be implemented in accordance with requirements outlined in the MIDP 2.0 chapter.

If the user selects not to launch the new MMS application then the incoming MMS message shall be ignored and deleted from the handset.

Application ending

At Application exit, WMA should remove all buffered messages that were not received by the Application.

If the MMS application needs to keep messages more persistently, it has to use other APIs (File System API, RMS, etc.) to save incoming MMS messages on the handset for later use. This is handled by the Application and outside the of scope of this MRS.

MMS Push

When received a message which has an unknown Application-ID, the MMS Engine shall validate the routing options registered by each of its clients.

Since the Application-ID does not match with the routing parameters, a NotifyRe-

sponse shall be sent back to the MMSC, with status set to REJECTED.

When received a message which has an unknown Application-ID / from_address, MMS Engine shall validate the routing options registered by each of its clients.

Since the Application-ID / from_address in the message does not match the routing parameters registered by the client, the handling of the message shall use one of the following methods:

- Retain the message in the native clients Inbox, or
- Delete the message and display the transient notice about the removal of the message.

The transient notice shall have the following UI dialogue elements (to be used for P04.4 integration releases):

- Generic Dialog
- Title Bar text: "Unknown message"
- Body Area text: "Message <message_index>was sent by an unknown application. Message deleted."
- Left Soft Key: [empty]
- Right Soft Key: OK
- The icon used in the Dialog: notice_generic_prmicn

Starting with P05.1 releases, due to entire UI adjustment for the 2-softkey paradigm, the "OK" shall be on the Left Soft Key.

The decisions to deploy options above shall be controlled by a Feature-ID as it depends on the behaviour desired by the Operator.

15.3 Requirements to the Native MMS Client

The mobile originated MMS Messages that are sent out by the Java client shall not be stored in the Native clients Draft/Outbox folder.

The Current OMA Standards do not support having the Application-ID parameter in the Notification. The Application-ID is only present in the Message body. Due to this, certain limitations exist as described here.

15.3.1 Anonymous Rejection Feature

The Native MMS Client shall support anonymous rejection feature.

When the MMS receives a Notification, if the from_address is not present in the notification, the message shall not be downloaded. A Notify Response shall be sent to the MMSC with the status set to REJECTED.

Filtering of the from_address shall be done at the Notification-Level.

(The impact of this scenario is that a message from an anonymous sender intended for the Java client will not be downloaded onto the handset.)

15.3.2 Coincidental Addresses in the native client and Java clients address filters

The Native MMS client shall maintain a black (Reject) list of address-filters.

Messages received with these addresses shall be rejected.

The Java client shall maintain an Acceptable list of address-filters: Only Messages that match this Address-filter shall be handled by the Java client.

Address-filtering shall be done at the Notification-level. If a message's from_address matches both the Native client and Java client's address-filters, the message shall not be downloaded and a Notify Response shall be sent to the MMSC with status set to REJECTED.

(So a message with this from_address, intended for the Java client will not be downloaded onto the handset.)

15.3.3 Security Policy

The WMA shall follow the security policy specified in MIDP 2.0 chapter.

To send and receive messages using WMA, applications shall be granted permission to perform the requested operation. The following table assigns individual permis-

sions:

Permission	Protocol	Function
javax.microedition.io.Connector.mms	mms	Connector.open("mms://....")
javax.wireless.messaging.mms.receive	mms	MessageConnection.setMessage-Listener MessageConnection.receive
javax.wireless.messaging.mms.send	mms	MessageConnection.send

When opening a connection, if the permission is not granted, then Connector.open method shall throw a SecurityException.

When sending or receiving messages, if the permission is not granted then the MessageConnection.send and the MessageConnection.receive methods shall throw a SecurityException.

15.3.4 VMVM support

WMA functionality shall be supported in VMVM environment.

15.3.5 External Event Interaction

The implementation shall follow external event interactions.

16 Java ME™ Access to certificates on SIM and phone memory

This chapter presents the specification to access digital certificates on "SIM or phone memory" by a Java Virtual Machine (JVM). The devices that support trusted applications must follow a PKI based authentication scheme as defined in MIDP 2.0 specification.

NOTE: Support of certificates on SIM cards, and the ability to delete a TTP certificate, are optional facilities whose availability may vary by region and carrier.

16.1 Allow JVM to access Digital Certificates

The following are the rules for accessing Digital certificates related to various domains Manufacturer, Operator and Trusted third party.

Rules:

- JVM must be able to read digital certificates in the SIM.
- JVM must be able to read digital certificates in the phone memory.
- The implementation MUST support the following certificates.
  - Manufacturer Domain
    - The certificate must be mapped to a secure location in the phone

memory.
    - If the certificate is not available on the device, the manufacturer domain MUST be disabled.
    - The certificate can only be deleted or modified by the manufacturer.
    - Any new or updated manufacturer protection domain root certificate must be associated with the manufacturer domain security policy on the device. MIDlet suites verified by a previous manufacturer protection domain root certificate MUST be disabled.
  - Operator Domain
  - The certificate must be mapped to the specified location in the SIM or in the phone memory.
    - If the certificate is not available on the specified location in the SIM or in the phone memory, the operator domain MUST be disabled.
    - The implementation MUST search SIM first for the operator root certificate.
    - The operator domain can't be deleted or modified by the application or any other party, except by device provisioned capability.
    - Number of "Operator domain" certificates to be stored in the SIM or in the phone memory should be at least 5.
  - Trusted third party Domain
    - The certificate must be mapped to the specified location in the SIM or in the phone memory.
    - Only operator can provision trusted third party root certificates in SIM. The certificate shall be stored as READ_ONLY. User/application can't disable/enable/delete trusted third party certificates stored in SIM.
    - The implementation must search SIM first for trusted third party root certificate.
    - If a certificate is not available at the specified location in the SIM or in the phone memory, the trusted third party domain must be disabled.
    - The user must not be able to delete or disable trusted third party protection domain root certificates which are stored as READ_ONLY.
    - Disabled trusted third party protection domain root certificates must not be used to verify downloaded MIDlet suites.
    - If this certificate is to be deleted, the user MUST be prompted to warn of the consequence of this action. This prompt MUST work in conjunction with the browser functionality.
    - If deleted or disabled, the third party domain MUST no longer be associated with this certificate.

    - Number of "Trusted third party protection domain" certificates to be stored in the SIM and phone memory should be at least 6.
- The implementation should cache SIM certificates in the phone memory at power up to avoid delay in retrieving certificates from SIM during installation/launching of a MIDlet.
- All the data in the certificate MUST be encoded using ASN.1/DER standards as specified in X.509.
- The implementation should follow the SIM interface specified in standard 3GPP TS 51.011.

16.2 Update certificates on the SIM

If the operator wants to issue or update the certificates on the SIM, they shall use the SIM Application Toolkit (SAT)to send the SIM Specific SMS, GPRS or Cell Broadcast message from the server to the SIM card for the certificate provisioning. Then the SIM Toolkit Refresh command shall be used to cause the certificates to be re-cached in the SIM card.

Rules:

- The operator shall issue or update the "Operator Root Certificate" and Trusted Third Party root certificates (SIM only) through the SIM Application ToolKit. The SIM Specific SMS, GPRS or Cell Broadcast message shall be sent from the server to the SIM card to remotely provision the certificates.
- The handset shall support the SIM Toolkit Refresh command which will be used to cause the certificate to be re-cached in the SIM card.

16.3 Procedure for viewing/enabling/deleting/disabling a certificate

The menu to view, enable/disable certificates is already implemented by the current Synergy design. The name of the menu is Certificates and can be accessed from both the Browser menu and the Security sub-menu under the main menu.

Currently, user is not allowed to delete a root certificate. This shall be changed to

allow certificates that are used by JVM to be deleted if needed. The user may view more detailed information about a particular certificate by highlighting the certificate name and selecting the VIEW soft key. Once selected, an "Editable List" will be created with options: Name, Expiration Date and Status(Enable/Disable). A new option: Delete must be added to this "Edit List" to allow it to be deleted.

- A new List Item "Delete" must be added to the "Editable List", if this feature is flexed on. It allows the user to select this item to delete the certificate. The procedure is shown in Figure 9 .
  - If the certificate is a Java type of certificate, the "Delete" item shows up in the "Editable List" only when the certificate belongs to the Trusted Third Party Domain and is not READ_ONLY. "Delete" item shall not be displayed for certificates stored as READ_ONLY.
  - If the certificate is a SSL type of certificate, the "Delete" item shall not be displayed.
- Enable/Disable/Delete options must not appear for Operator and Manufacturer Domain Root Certificates.
- Enable/Disable options must appear for Trusted Third Party certificates which are not READ_ONLY. Enable/Disable options must not appear for certificates stored as READ_ONLY.

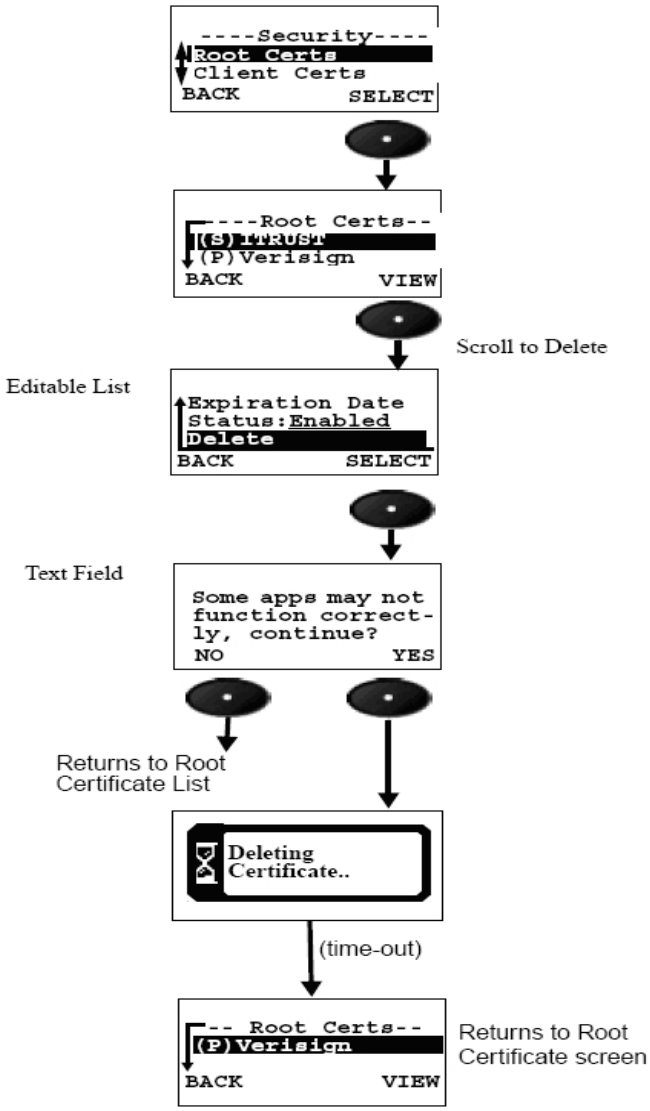


Figure 9 Delete a Trusted Third Party Domain Root Certificate

The menu to view, enable/disable certificates is already implemented by the current Synergy design. The name of the menu is Certificates and can be accessed from both the Browser menu and the Security sub-menu under the main menu.

Currently, user is not allowed to delete a root certificate. The user may view more detailed information about a particular certificate by highlighting the certificate name and selecting the VIEW soft key. Once selected, an "Editable List" will be created with options: Name, Expiration Date and Status(Enable/Disable).

16.4 Roaming/Change of SIM card

All previously authorized and installed MIDlets MUST act in accordance with the device policy when the device is roaming, or when the device SIM is changed. Newly downloaded MIDlets are authenticated to the root certificates currently available in the certificate store and authorized in accordance with the device policy.

If device roaming or a SIM card change causes failure to access network resources that the MIDlet was previously authorized to access, then the implementation MUST throw an IOException rather than security exception.

17 Prevent Downloading of Large Java MIDlets

17.1 Overview

This feature is a 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 must be the 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 "Java ME™ 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 the 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.

17.2 Notification

When the JAR file size exceeds the maximum value set, a 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.

17.3 Backward Compatibility/Flexing

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

18 Download Midlet through PC

To download MIDlets through a PC, make a connection to a PC through IrDA, Bluetooth, USB or Serial Cable (RS 232). This content considers only the RS232 connection using JAL.

18.1 Establishing Connection

When a successful connection to a PC is made, an application can be downloaded. The MS should display that a connection has been made. Only one connection will be active at a time.

19 Downloading MIDlet through Browser

The Download MIDlet Through Browser requires the browser to be connected before performing any downloads on the handset.

The example shows How user may access the Browser application by any of the following methods:

- Selecting 'Browser' from the Main Menu.
- Pressing a dedicated 'Browser' key on the keypad (if available on the handset).
- Pressing a 'Browser' soft key from the idle display (if assigned).
- Using 'Browser' shortcut (if assigned).
- Selecting URL from a message.
- Selecting GetJavaApps from the Main Menu or Java Settings.

19.1 Star Active Browser Session from Main Menu

Figure 10 describes Staring Active Browser Session from Main Menu:

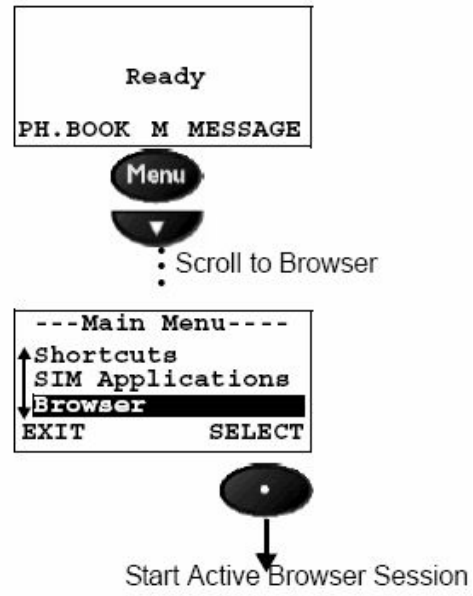


Figure 10 Starting Active Browser Session from Main Menu

GetJavaApps is a feature that allows an operator to insert a WAP designated URL that links to a Java ME ™ site with MIDlet suites. This feature can be found under Java Settings or in the Main Menu as it is flexible for either menu item.

19.2 Find a location with Java ME ™ Application

Once connected to the WAP browser, different locations may be visited where Java ME ™ Applications may be downloaded. From here, a MIDlet may be selected to download to the handset.

Handset initially receives information from the Java Application Descriptor (JAD) file. The JAD includes information about MIDlet-name, version, vendor, MIDlet-Jar-URL, MIDlet-Jar-size, and MIDlet-Data-size. Two additional JAD attributes will be Mot-Data-Space-Requirements and Mot-Program-Space-Requirements. These attributes will help the KVM determine whether there is enough memory to download and install the selected MIDlet suite. If there is not enough memory, 'Memory Full' dialog will be displayed before the download begins.

19.3 Downloading MIDlets

Figure 11 represents Java ME ™ Application (MIDlets) Download and Installation.

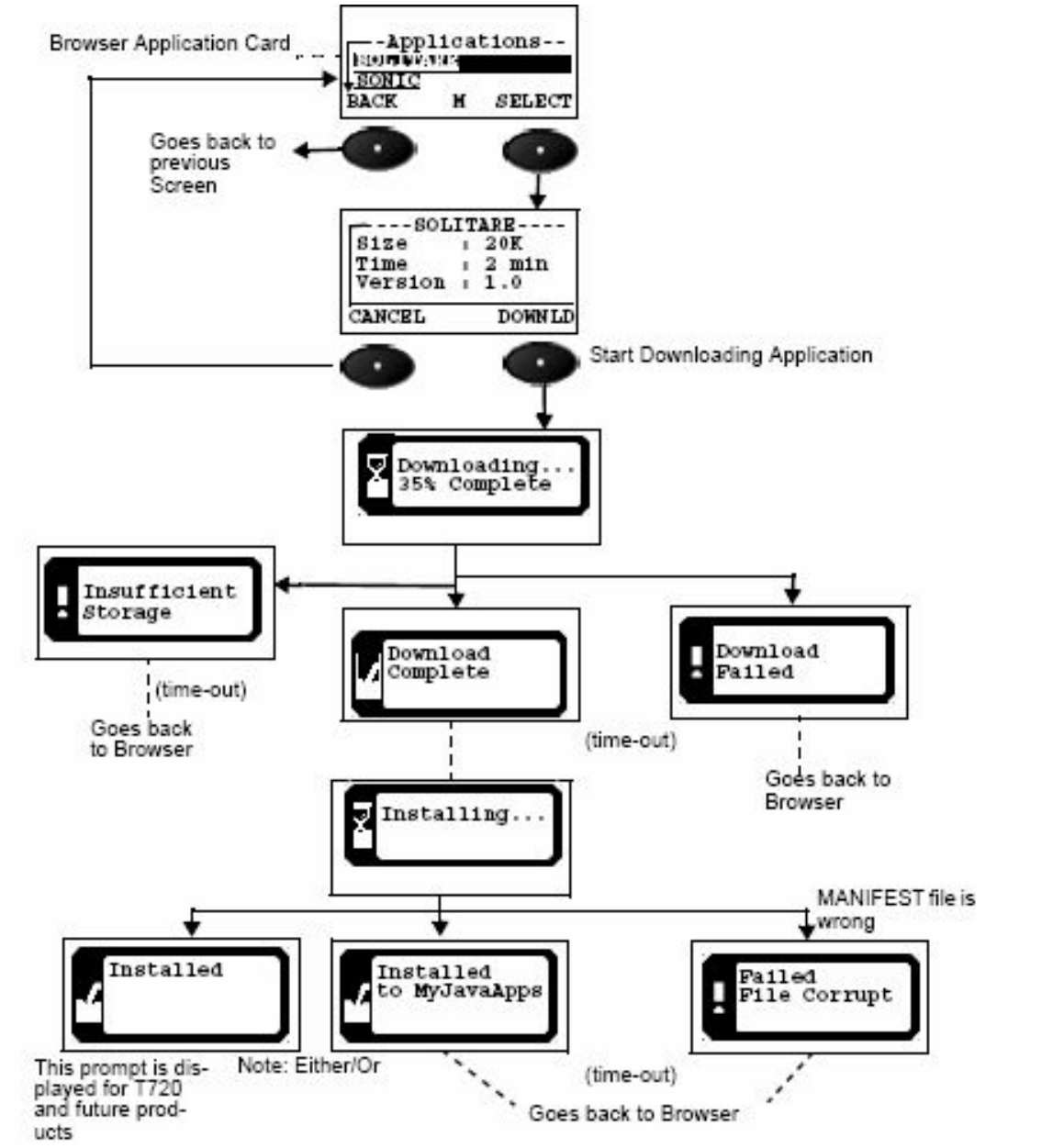


Figure 11 Downloading and Installing Java ME ™ Application (MIDlets)

Steps to Download and Install Java ME ™ Application:

- BACK shows previous screen to the user.
- If the SELECT softkey is selected, the handset shows display the application size, time to install and version. If an error occurs with the descriptor file, the handset then displays the transient notice 'Failed Invalid File.' Upon Time-out, the handset goes back to browser.
- If the CANCEL softkey is selected, it shows the Browser Application Card from where the application was selected.
- If the DOWNLD softkey is selected, the handset starts downloading the application. The handset displays 'Downloading...% Complete' along with the percentage of downloading completed at a time. 'Downloading...% Complete' shall use static dots, not dynamic.
- Before downloading the MIDlet, handset checks for available memory. Mot-Data-Space-Requirements and Mot-Program-Space-Requirements are two JAD attributes that will help the KVM determine whether there is enough memory to download and install the selected MIDlet suite. If there is not enough memory, 'Insufficient storage' transient dialog will be displayed before the download begins. Upon time-out, the handset goes back to browser.
- If an error occurs during download, such as a loss of service, then the transient notice 'Download Failed' must be displayed. Upon time-out, the handset goes back to idle state.
- A downloading application can be cancelled by pressing the END key. The transient notice, 'Download Cancelled, ' displays. Upon time-out, handset goes back to browser.
- 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 the downloading application is cancelled, handset cleans up all files, including any partial JAR files and temporary files created during the download process.
- When downloading is done, the handset displays a transient notice 'Download Completed'. The handset then starts to install the application.
- The handset displays 'Installing...'.
- After an application is successfully downloaded, a status message must be sent back to the network server. This allows for charging of the downloaded application.
- Charging is per the Over the Air User Initiated Provisioning specification. The status of an install is reported by means of an HTTP POST request to the URL contained in the MIDlet-Install-Notify attribute. The only protocol that MUST be supported is 'http://'.
- If the browser connection is interrupted/ended during the download/installation process, the device will be unable to send the HTTP POST with the MIDlet-Install Notify attribute. In this case, the MIDlet will be deleted to insure the user does not get a free MIDlet. The use case can occur when a phone call is accepted and terminated during the installation process, because then the browser will not be in the

needed state in order to return the MIDlet Install Notify attribute.

- Upon completing Installation, the handset displays a transient notice 'Installed to Games & Apps'.
- Upon time-out, the handset goes back to Browser.
- During Installation if the MANIFEST file is wrong, the handset displays a transient notice 'Failed File Corrupt'. Upon time-out, the handset goes back to Browser.
- During the installation process, if the flip is closed on a flip handset, the installation process will continue and the handset will not return to the idle display. When the flip is opened, the 'Installing...' dialogue should appear on the screen and should be dynamic.
- During download or install of application, voice record, voice commands, voice shortcuts, and volume control will not be supported. However, during this time, incoming calls and SMS messages are able to be received.
- The handset must support sending and receiving at least 30 kilobytes of data using HTTP either from the server to the client or the client to the server, per Over the Air User Initiated Provisioning specification.
- If JAD does not contain mandatory attributes, 'Failed Invalid File' notice appears.

If JAD does not contain mandatory attributes, 'Failed Invalid File' notice appears.

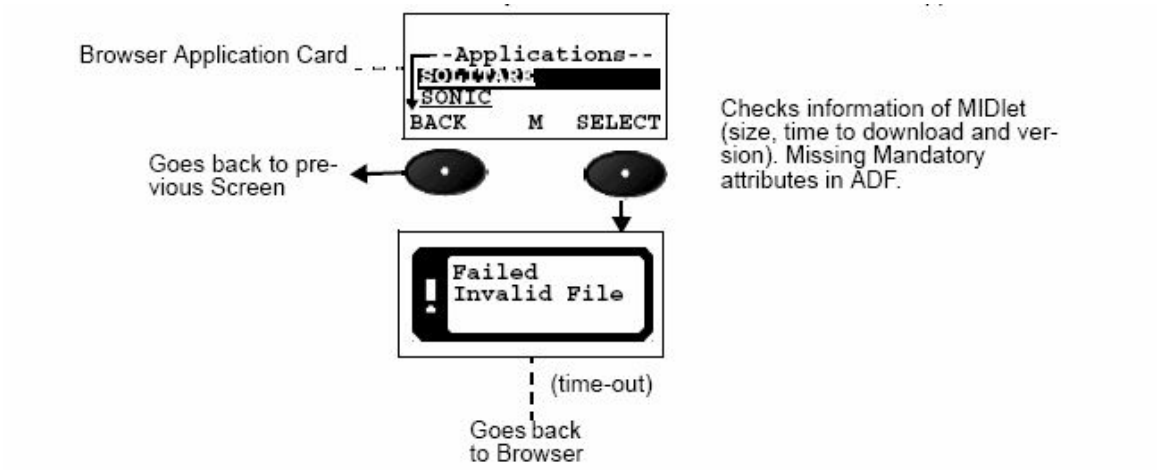


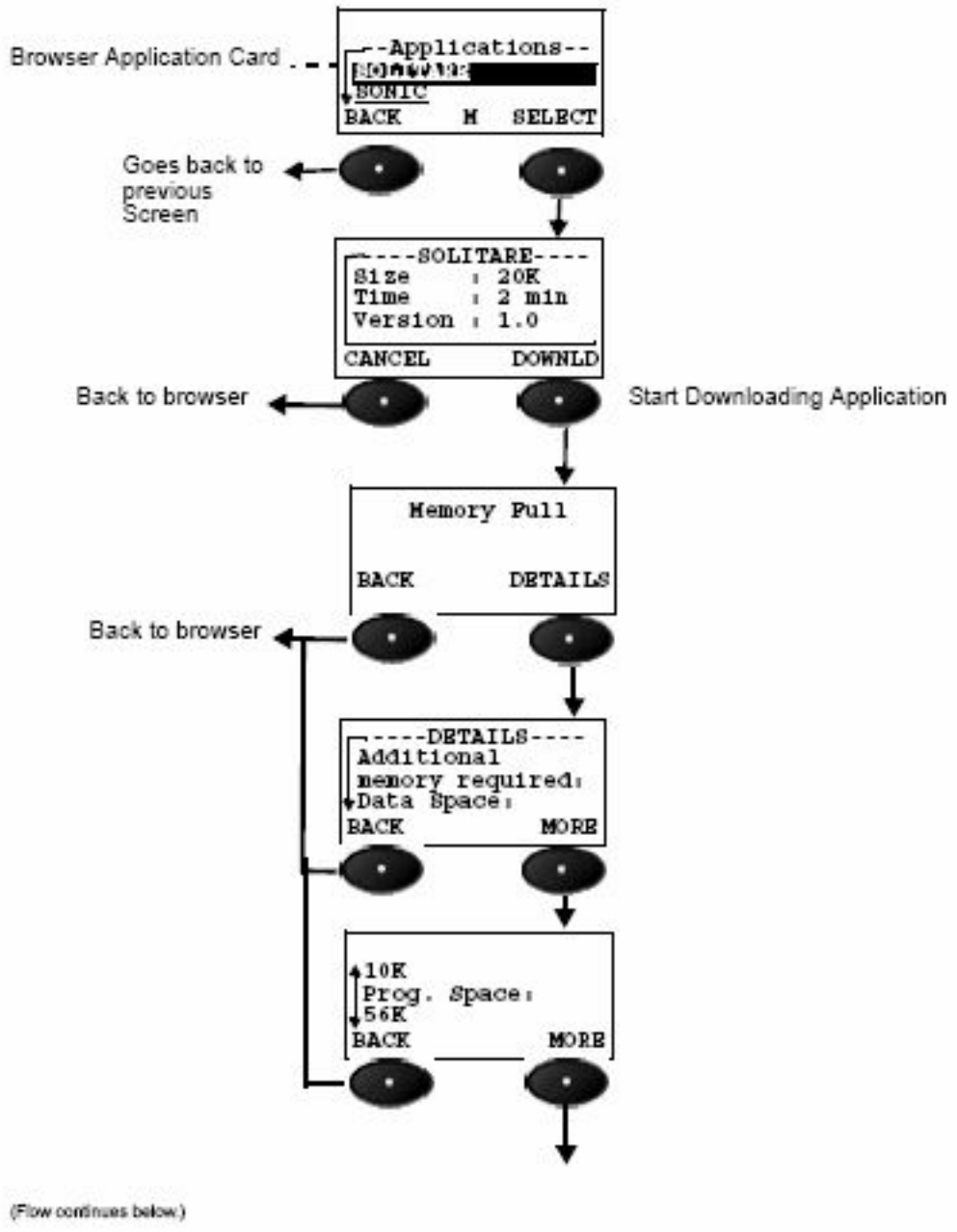
Figure 12 Application does not have Mandatory Attributes in ADF

19.4 Different Error Checks

19.4.1 Memory Full

There are two distinct cases when a Memory Full error can occur during the download process. Memory Full will be displayed when the device does not have enough memory to completely download the MIDlet. The JAD of the MIDlet has two attributes, Mot-Data-Space-Requirements and Mot-Program-Space-Requirements. If an application developer adds these attributes to their JAD file, a Motorola device can determine if enough memory exists on the phone before the MIDlet is downloaded. These attributes may or may not be provided in all MIDlets. Two separate prompts will be displayed depending on whether these attributes are present.

In cases where there is not enough memory to download the application, the user MUST be given a message to delete existing applications in order to free additional memory. The following messages and screen flows will be displayed depending on whether specific JAD attributes are present or not:


Browser Application Card
--Applications--
SONIC
BACK
M
SELECT
Goes back to previous Screen
----SOLITARE----
Size : 20K
Time : 2 min
Version : 1.0
CANCEL
DOWNLD
Back to browser
Start Downloading Application
Memory Full
BACK
DETAILS
Back to browser
----DETAILS----
Additional memory required:
Data Space:
BACK
MORE
10K
Prog. Space:
56K
BACK
MORE
(Flow continues below)

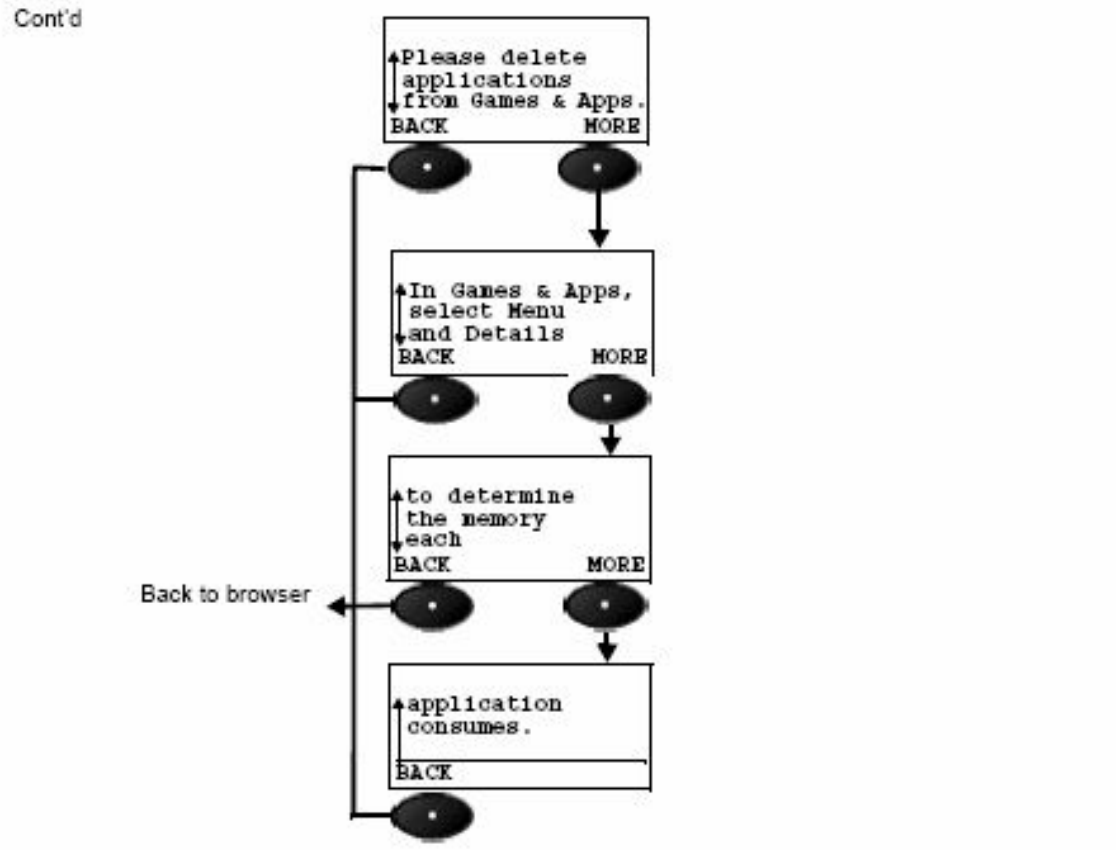


Figure 13 Memory full error

Rules:

- If Mot-Data-Space-Requirements and Mot-Program-Space-Requirements attributes are present in the JAD, the above noted prompt should be displayed. This value takes into account the memory requirements of the MIDlet and the current memory usage on the phone, in order to tell the user exactly how much memory to free. The memory usage is based in kilobyte units.
- 'Data Space:' and the value of the data space should be on separate lines. 'Prog. Space:' and the value of the program space should be on separate lines.
- The download process is canceled when this error condition occurs.
- The Memory Full error will no longer be a transient prompt but a dialog screen with a Help softkey and a Back softkey will be displayed.
- DETAILS will give the user the above detailed Help screen describing the memory required to be able to download the MIDlet.
- The Help dialog will include a 'More' right softkey label (for those products in which not all the help data can be displayed on a single screen). This label should disappear when the user has scrolled to the

bottom of the dialog.
- BACK from this message will take the user back to the browser page from which the user selected the MIDlet to download.

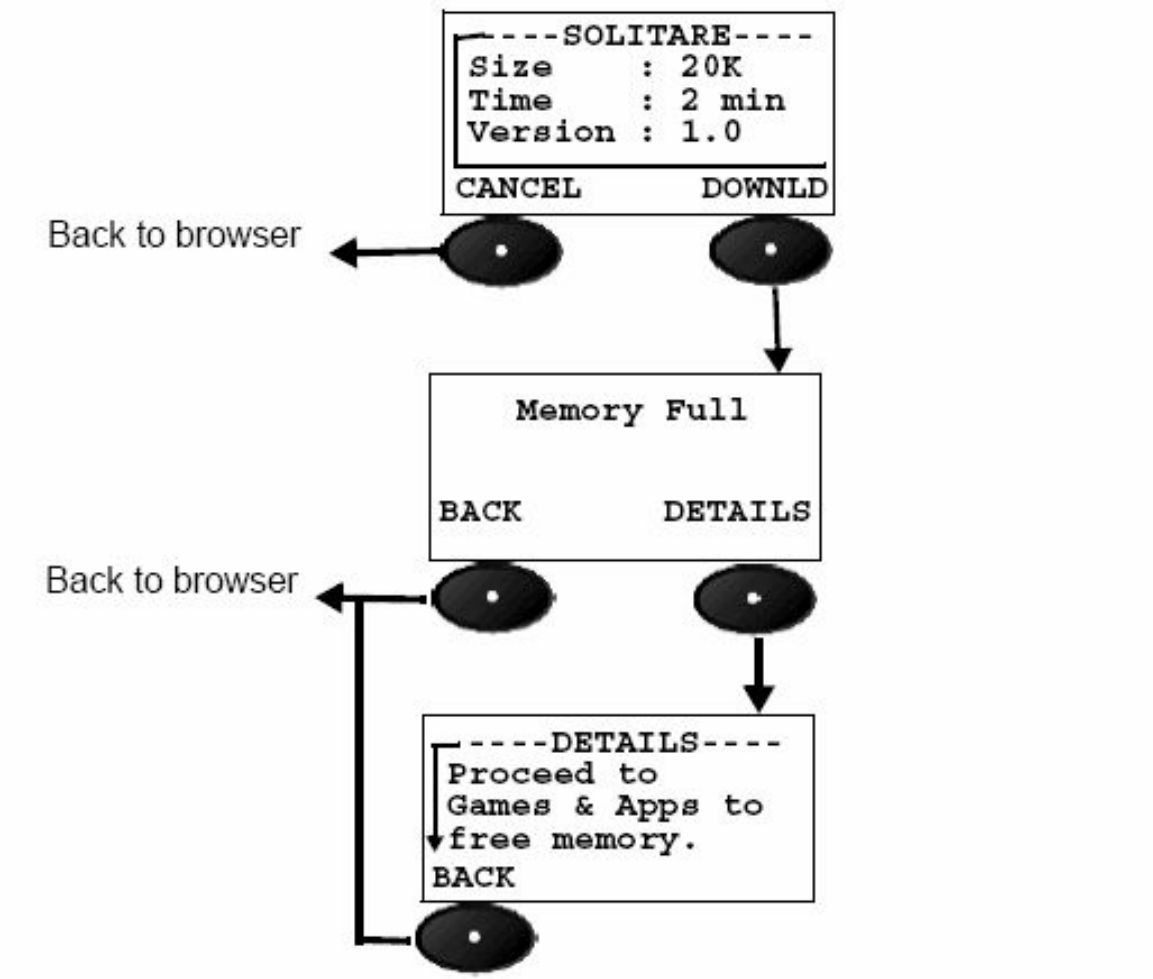


Figure 14 Mot-Data-Space & Mot-Program-Space attributes are not present or are incorrect

- If Mot-Data-Space-Requirements and Mot-Program-Space-Requirements JAD attributes are not present in the JAD file, the handset can not determine how much memory to free and will display the above help dialog.
- The Help dialog will include a 'More' right softkey label (for those products in which not all the help data can be displayed on a single screen). This label should disappear when the user has scrolled to the bottom of the dialog.
- All rules stated in the previous figure must also be followed for the above stated prompt.

19.4.2 Memory Full during installation process.

Once the MIDlet is successfully downloaded, the installation process begins. During the installation of the MIDlet, the phone may determine there is insufficient memory to complete the installation. This error can occur whether the Mot-Data-Space-Requirements and Mot-Program-Space-Requirements JAD attributes are present or not. The following message and Figure Figure 15 must be displayed:

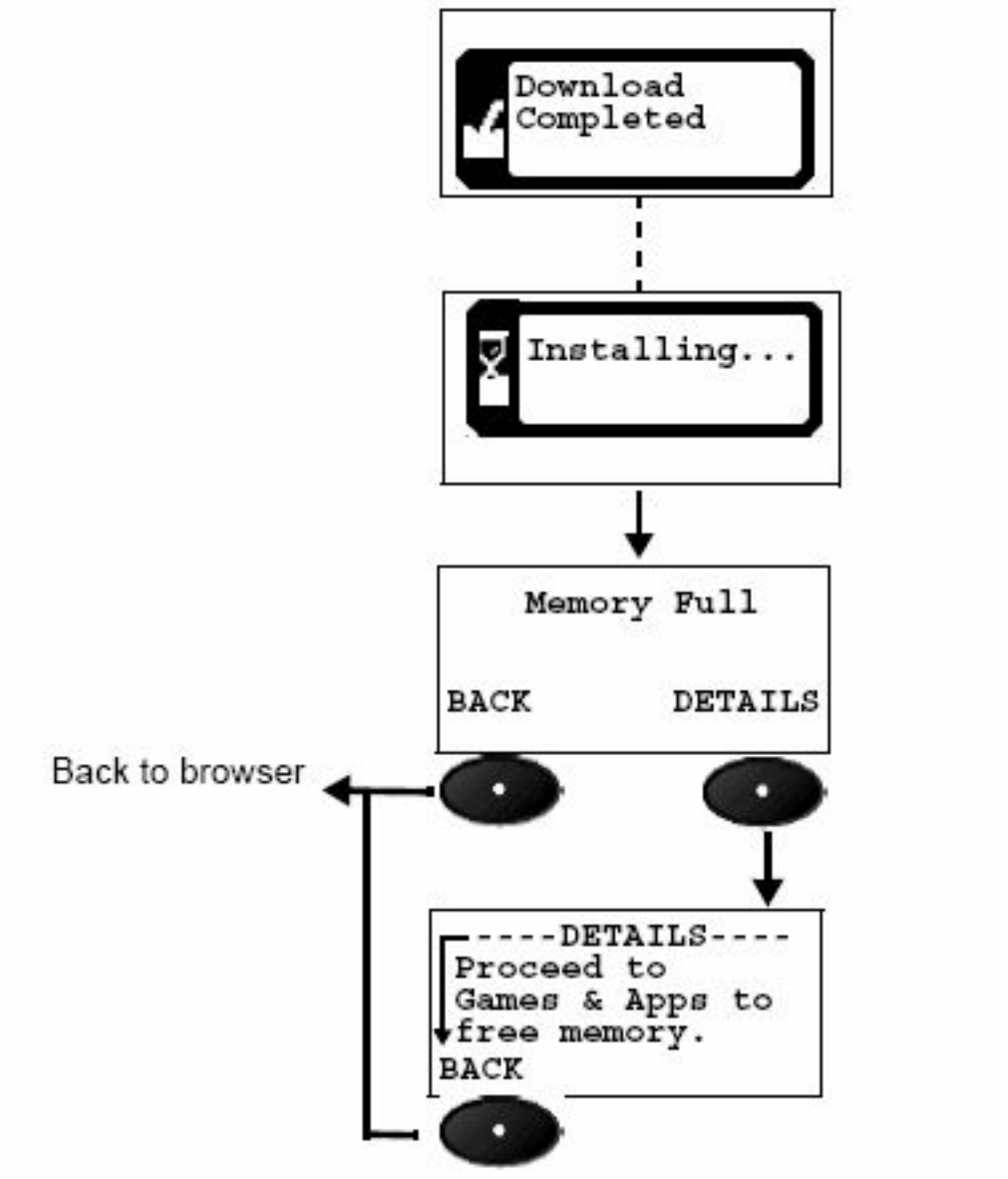


Figure 15 Memory Full help message during installation process

- The installation process is canceled when this error condition occurs.
- The Memory Full error will no longer be a transient prompt but a dialog screen with a Help softkey and a Back softkey will be displayed.
- DETAILS will give the user the above Help screen explaining that additional memory is required to be able to install the MIDlet.
- The Help dialog will include a 'More' right softkey label (for those products in which not all the help data can be displayed on a single screen). This label should disappear when the user has scrolled to the bottom of the dialog.
- BACK from this message will take the user back to the browser page from which the user selected the MIDlet to download.

19.4.3 Application version already exists

Compares the version number of the application with that already present on the handset. If the versions are the same, the following message is displayed. The error occurred can be queried by selecting DETAILS.

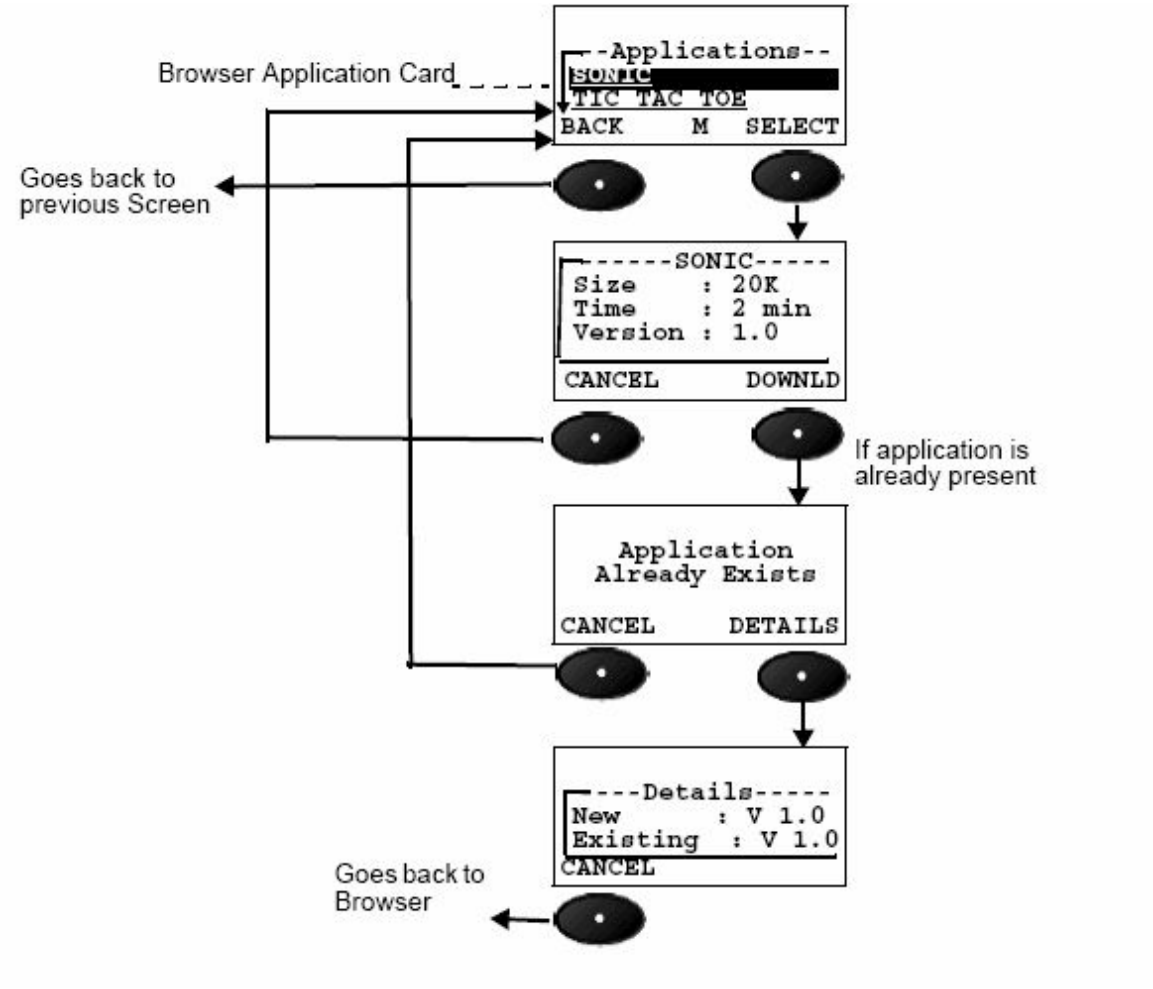


Figure 16 Same Version of Application already exists on the handset

Rules:

- Handset checks for MIDlet-Name, MIDlet-vendor, and version number. If they are the same, a dialog 'Application Already Exists' is displayed.
- To know more about this error, select the DETAILS softkey.
- Handset displays the new version of the application, as well as the existing application.

19.4.4 Newer application version exists

If the application version on the handset is newer than the downloaded version of

application, the following message is displayed. The error occurred can be queried by selecting DETAILS.

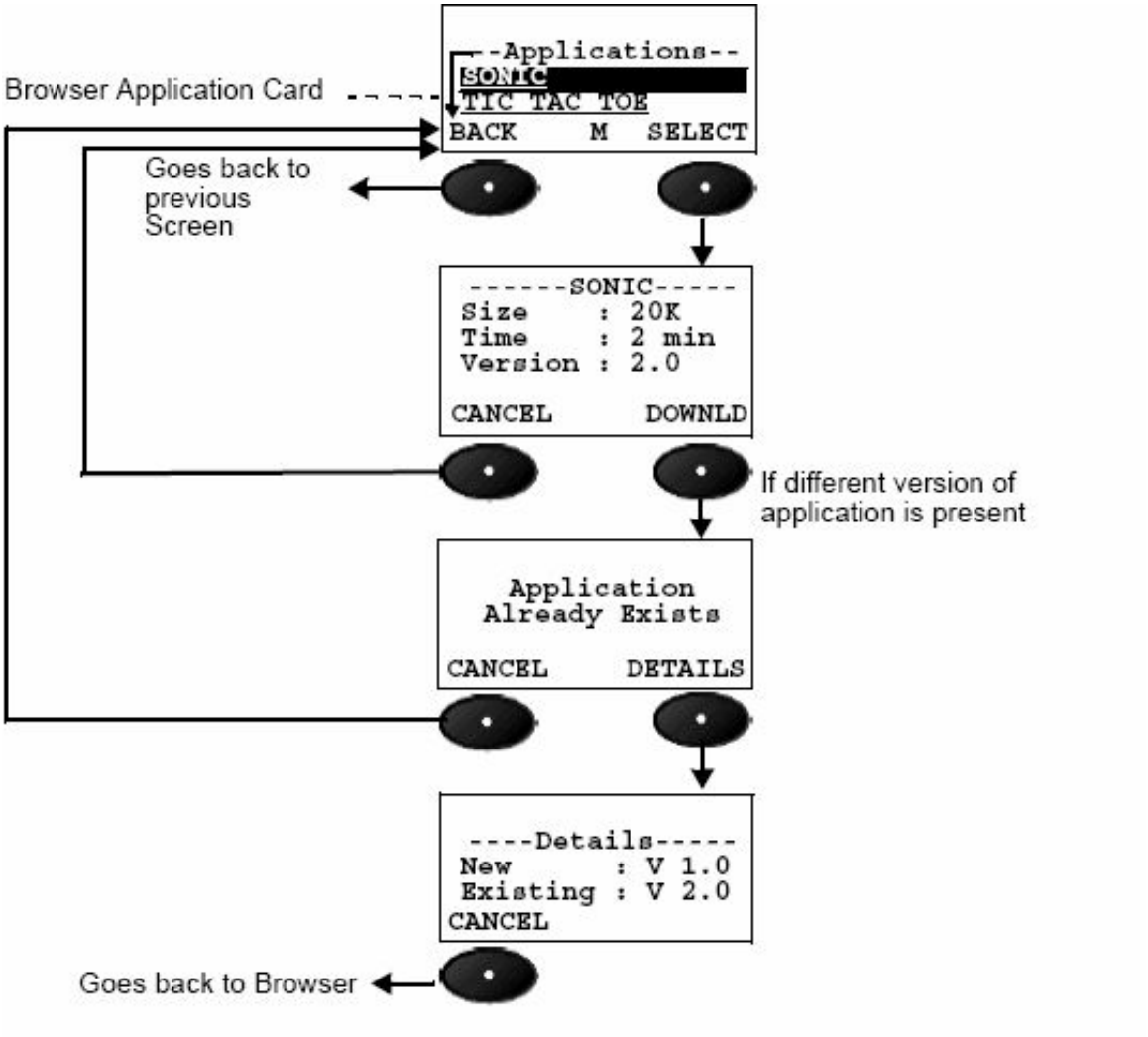


Figure 17 (Newer) Version of Application exists

Rules:

- If the latest or newer version of application is already present on the handset, it cannot be downloaded.

20 Record Management System

20.1 Record Management System API

If the MIDlet has not specified a data space requirement in the JAD attribute (MIDlet_data_space_requirement) or the manifest file, a limit of 512 KB will be used as the maximum RMS space for that MIDlet. No additional Motorola implementation clarifications are necessary.

Refer to Table 27 for RMS feature/class support for MIDP 2.0:

Feature/Class	Implementation
All constructors, methods, and inherited methods for the InvalidRecordDException class in the javax.microedition.rms package	Supported
All fields and methods for the RecordComparator interface in the javax.microedition.rms package	Supported
All methods for the RecordEnumeration interface in the javax.microedition.rms package	Supported
All methods for the RecordFilter interface in the javax.microedtition.rms package	Supported
All methods for the RecordListener interface in the javax.microedition.rms package	Supported
All fields, methods, and inherited methods fortify the RecordStore interface in the javax.microedition.rms package	Supported
Initial version number of a record to be zero	Supported
All constructors, methods, and inherited methods for the RecordStoreException class in the	Supported

javax.microedition.rms package	
All constructors, methods, and inherited methods for the RecordStoreFullException class in the javax.microedition.rms package	Supported
All constructors, methods, and inherited methods for the RecordStoreNotOpenException class in the javax.microedition.rms package	Supported
All constructors, methods, and inherited methods for the InvalidRecordIDException class in the javax.microedition.rms package	Supported
All fields and methods for the RecordComparator interface in the javax.microedition.rms package	Supported
All methods for the RecordEnumeration interface in the javax.microedition.rms package	Supported
All methods for the RecordFilter interface in the javax.microedition.rms package	Supported
All methods for the RecordListener interface in the javax.microedition.rms package	Supported
All fields, methods, and inherited methods for the RecordStore interface in the javax.microedition.rms package	Supported
All constructors, methods, and inherited methods for the RecordStoreException class in the javax.microedition.rms package	Supported
All constructors, methods, and inherited methods for the RecordStoreNotFoundException class in the javax.microedition.rms package	Supported
All constructors, methods, and inherited methods for the RecordStoreNotOpenException class in the javax.microedition.rms package	Supported

Table 27 RMS feature/class

21 Gaming API/Multiple Key Press

21.1 Gaming API

The Gaming API provides a series of classes that enable rich gaming content for the handset. This API improves performance by minimizing the amount of work done in Java, decreasing application size. The Gaming API is structured to provide freedom in implementation, extensively using native code, hardware acceleration, and device-specific image data formats as needed.

The API uses standard low-level graphic classes from MIDP so the high-level Gaming API classes can be used in conjunction with graphics primitives. This allows for rendering a complex background using the Gaming API while rendering something on top of it using graphics primitives.

Methods that modify the state of Layer, LayerManager, Sprite, and TiledLayer objects generally do not have any immediate visible side effects. Instead, this state is stored within the object and is used during subsequent calls to the `paint()` method. This approach is suitable for gaming applications where there is a cycle within the objects' states being updated and the entire screen is redrawn at the end of every game cycle.

21.2 Multiple Key Press Support

Multi-button press support enhances the gaming experience for the user. Multi-button press support gives the user the ability to press two (2) keys simultaneously and the corresponding actions of both keys will occur simultaneously. An example of this action would be the following:

- Simultaneously moving to the right and firing at objects in a game.

The following sets of keys will support multi-button press support on the MOTORAZR V3xx handset. Multi-button press within each set will be supported, while multi-button press across these sets or with other keys will not be supported.

Set 1 - Nav (Up), Nav (Down), Nav (Right), Nav (Left), 9

Set 2 - 2, 4, 6, 8, 7

Set 3 - 0, #

Refer to Table 28 for gaming and keypad feature/class support for MIDP 2.0:

Feature/Class	Implementation
lcdui.game package	Supported
setBacklight as defined in javax.microedition.lcdui.Display	Supported
setVibrator as defined in javax.microedition.lcdui.Display	Supported
All constructors and inherited classes for the IllegalStateException in java.lang	Supported
All constructors, methods, and inherited classes for the Timer class in java.util	Supported
All the constructors, methods, and inherited classes for the TimerTask class in java.util	Supported
All fields, constructors, methods, inherited fields and inherited methods for the GameCanvas class in javax.microedition.lcdui.game	Supported
GameCanvas size	9x larger than screen
Map the UP_PRESSED field in javax.microedition.lcdui.game.GameCanvas to the top position of the key.	Supported
Map the DOWN_PRESSED field in	Supported

javax.microedition.lcdui.GameCanvas to the bottom position of the key	
Map the LEFT_PRESSED field in javax.microedition.lcdui.GameCanvas to the left position of the key	Supported
Map the RIGHT_PRESSED field in javax.microedition.lcdui.GameCanvas to the right position of the key	Supported
All methods and inherited methods for the Layer class in javax.microedition.lcdui.game	Supported
All constructors, methods, and inherited methods for the LayerManager class in javax.microedition.lcdui.game.Layer	Supported
All fields, constructors, methods, and inherited methods for the Sprite Class in javax.microedition.lcdui.game	Supported
Sprite Frame height will not be allowed to exceed the height of the view window in javax.microedition.lcdui.Layer	Any, limited by heap size only
Sprite frame width will not be allowed to exceed the width view of the view window in javax.microedition.lcdui.Layer	Any, limited by heap size only
Sprite recommended size	16*16 or 32*32
All constructors, methods, and inherited methods for the TiledLayer class in javax.microedition.lcdui.game	Supported
MIDlet Queries to keypad hardware	Supported
Alpha Blending	Supported

Table 28 Gaming and keypad feature/class support for MIDP

22 Network APIs

22.1 Network Connections

The Motorola implementation of Networking APIs will support several network connections. The network connections necessary for Motorola implementation are the following:

- CommConnection for serial interface
- HTTP connection
- HTTPS connection
- Push registry
- SSL (secure socket)
- Datagram (UDP)

Refer to Table 29 for Network API feature/class support for MIDP 2.0:

Feature/Class	Implementation
All fields, methods, and inherited methods for the Connector class in the javax.microedition.io package	Supported
Mode parameter for the open () method in the Connector class the javax.microedition.io package	READ, WRITE, READ_WRITE
The timeouts parameter for the open () method in the Connector class of the javax.microedition.io package	
HttpConnection interface in the javax.microedition.io package	Supported
HttpsConnection interface in the javax.microedition.io package	Supported
SecureConnection interface in the javax.microedition.io package	Supported
SecurityInfo interface in the javax.microedition.io package	Supported
UDPDDatagramConnection interface in the javax.microedition.io package	Supported
Connector class in the javax.microedition.io.package	Supported
PushRegistry class in the javax.microedition.io package	Supported

CommConnection interface in the javax.microedition.io package	Supported
Dynamic DNS allocation through DHCP	Supported
HttpConnection interface in the javax.microedition.io.package.	Supported
HttpsConnection interface in the javaxmicroedition.io.package	Supported
SecureConnection interface in the javax.microedition.io.package	Supported
SecurityInfo Interface in the javax.microedition.io.package	Supported
UDPDatagramConnection interface in the javax.microedition.io.package	Supported

Table 29 Network API feature/class support for MIDP

Code Sample 8 shows the implementation of Socket Connection:

Socket Connection

```
import javax.microedition.io.*;
import java.io.*;
import javax.microedition.midlet.*;

...

       try {
             //open the connection and io streams
                sc = (SocketConnection)Connector.open
("socket://www.myserver.com:8080", Connector.READ_WRITE, true);
                is = sc[i].openInputStream();
                os = sc[i].openOutputStream();

          } catch (Exception ex) {
             closeAllStreams();
             System.out.println("Open Failed: " + ex.getMessage());
           }
       }
       if (os != null && is != null)
       {
          try
          {
             os.write(someString.getBytes()); //write some data to server

             int bytes_read = 0;
             int offset = 0;
             int bytes_left = BUFFER_SIZE;

          //read data from server until done
```

```
            do
            {
               bytes_read = is.read(buffer, offset, bytes_left);

               if (bytes_read > 0)
               {
                  offset += bytes_read;
                  bytes_left -= bytes_read;
                }
            }
            while (bytes_read > 0);

         } catch (Exception ex) {
               System.out.println("IO failed: "+ ex.getMessage());
         }
         finally {
            closeAllStreams(i); //clean up
         }
      }
```

Code Sample 8 Socket Connection

22.2 User Permission

The user of the handset will explicitly grant permission to add additional network connections.

22.3 Indicating a Connection to the User

When the java implementation makes any of the additional network connections, it will indicate to the user that the handset is actively interacting with the network. To indicate this connection, the network icon will appear on the handset's status bar as shown in Figure 18 .

Figure 18 Network Connections example

Conversely, when the network connection is no longer used the network icon will be removed from the status bar.

If the handset supports applications that run when the flip is closed, the network icon on the external display will be activated when the application is in an active network connection with the flip closed. Please note that this indication is done by the implementation.

22.4 HTTPS Connection

Motorola implementation supports a HTTPS connection on the MOTORAZR V3xx handset. Additional protocols that will be supported are the following:

TLS protocol version 1.0 as defined in *http://www.ietf.org/rfc/rfc2246.txt*

SSL protocol version 3.0 as defined in *http://wp.netscape.com/eng/ssl3/ssl-toc.html*

Code Sample 9 shows the implementation of HTTPS:

HTTPS

```
import javax.microedition.io.*;
import java.io.*;
import javax.microedition.midlet.*;



      try {
         hc[i] = (HttpConnection)Connector.open("https://" + url[i] + "/");

      } catch (Exception ex) {
```

```
            hc[i] = null;
            System.out.println("Open Failed: " + ex.getMessage());
        }

        if (hc[i] != null)
        {
            try {
                is[i] = hc[i].openInputStream();

                 byteCounts[i] = 0;
                 readLengths[i] = hc[i].getLength();

                 System.out.println("readLengths = " + readLengths[i]);

                if (readLengths[i] == -1)
                {
                    readLengths[i] = BUFFER_SIZE;
                }

                int bytes_read = 0;
                int offset = 0;
                int bytes_left = (int)readLengths[i];

                do
                {
                    bytes_read = is[i].read(buffer, offset, bytes_left);
                    offset += bytes_read;
                    bytes_left -= bytes_read;
                    byteCounts[i] += bytes_read;
                }
                while (bytes_read > 0);

                System.out.println("byte read = " + byteCounts[i]);

                } catch (Exception ex) {
                        System.out.println("Downloading Failed: "+ ex.getMessage());
                        numPassed = 0;
                }
               finally {
                     try {
                             is[i].close();
                             is[i] = null;
                       } catch (Exception ex) {}
                }
        }
```

```
/**
 *  close http connection
 */
if (hc[i] != null)
{
   try {
      hc[i].close();
   } catch (Exception ex) { }
   hc[i] = null;
}
```

Code Sample 9 HTTPS

22.5 DNS IP

The DNS IP will be flexed on or off (per operator requirement) under Java Settings as read only or as user-editable. In some instances, it will be flexed with an operator-specified IP address.

22.6 Push Registry

The push registry mechanism allows an application to register for notification events that are meant for the application. The push registry maintains a list of inbound connections.

22.7 Mechanisms for Push

Motorola implementation for push requires the support of certain mechanisms. The mechanisms that will be supported for push are the following:

SMS push: an SMS with a port number associated with an application used to deliver the push notification.

The formats for registering any of the above mechanisms will follow those detailed in JSR-118 specification.

22.8 Push Registry Declaration

The application descriptor file will include information about static connections that are needed by the MIDlet suite. If all static push declarations in the application descriptor cannot be fulfilled during the installation, the MIDlet suite will not be installed. The user will be notified of any push registration conflicts despite the mechanism. This notification will accurately reflect the error that has occurred.

Push registration can fail as a result of an Invalid Descriptor. Syntax errors in the push attributes can cause a declaration error resulting in the MIDlet suite installation being cancelled. A declaration referencing a MIDlet class not listed in the MIDlet-<n> attributes of the same application descriptor will also result in an error and cancellation of the MIDlet installation.

Two types of registration mechanisms will be supported. The registration mechanisms to be supported are the following:

Registration during installation through the JAD file entry using a fixed port number

Dynamically register using an assigned port number

If the port number is not available on the handset, an installation failure notification will be displayed to the user while the error code 911 push is sent to the server. This error will cease the download of the application.

Applications that wish to register with a fixed port number will use the JAD file to identify the push parameters. The fixed port implementation will process the MIDlet-Push-n parameter through the JAD file.

Code Sample 10 shows the implementation of Push Registry:

Push Registry Declaration

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.PushRegistry;

public class PushTest_1 extends MIDlet implements CommandListener{

        public Display display;
```

```
    public static Form regForm;
    public static Form unregForm;
    public static Form mainForm;
    public static Form messageForm;

    public static Command exitCommand;
    public static Command backCommand;
    public static Command unregCommand;
    public static Command regCommand;

    public static TextField regConnection;
    public static TextField regFilter;
    public static ChoiceGroup registeredConnsCG;
    public static String[] registeredConns;

    public static Command mc;
    public static Displayable ms;

    public PushTest_1(){
        regConnection = new TextField("Connection port:", "1000", 32, Text-
Field.PHONENUMBER);
        regFilter = new TextField("Filter:", "*", 32, TextField.ANY);

        display = Display.getDisplay(this);

        regForm = new Form("Register");
        unregForm = new Form("Unregister");
        mainForm = new Form("PushTest_1");
        messageForm = new Form("PushTest_1");

        exitCommand = new Command("Exit", Command.EXIT, 0);
        backCommand = new Command("Back", Command.BACK, 0);
        unregCommand = new Command("Unreg", Command.ITEM, 1);
        regCommand = new Command("Reg", Command.ITEM, 1);

        mainForm.append("Press \"Reg\" softkey to register a new connection.\n" +
                "Press \"Unreg\" softkey to unregister a connection.");
        mainForm.addCommand(exitCommand);
        mainForm.addCommand(unregCommand);
        mainForm.addCommand(regCommand);
        mainForm.setCommandListener(this);

        regForm.append(regConnection);
        regForm.append(regFilter);
```

```
        regForm.addCommand(regCommand);
        regForm.addCommand(backCommand);
        regForm.setCommandListener(this);

        unregForm.addCommand(backCommand);
        unregForm.addCommand(unregCommand);
        unregForm.setCommandListener(this);

        messageForm.addCommand(backCommand);
        messageForm.setCommandListener(this);


    }
    public void pauseApp(){}


    protected void startApp() {
        display.setCurrent(mainForm);
    }

    public void destroyApp(boolean unconditional) {
        notifyDestroyed();
    }

    public void showMessage(String s) {
        if(messageForm.size() != 0 ) messageForm.delete(0);
        messageForm.append(s);
        display.setCurrent(messageForm);
    }

    public void commandAction(Command c, Displayable s) {

        if((c == unregCommand) && (s == mainForm)){
            mc = c;
            ms = s;
            new runThread().start();
        }

        if((c == regCommand) && (s == mainForm)){
            display.setCurrent(regForm);
        }

        if((c == regCommand) && (s == regForm)){
            mc = c;
            ms = s;
```

```
            new runThread().start();
         }

         if((c == unregCommand) && (s == unregForm)){
            mc = c;
            ms = s;
            new runThread().start();
         }

         if((c == backCommand) && (s == unregForm )){
            display.setCurrent(mainForm);
         }
         if((c == backCommand) && (s == regForm )){
            display.setCurrent(mainForm);
         }

         if((c == backCommand) && (s == messageForm)){
            display.setCurrent(mainForm);
         }

         if((c == exitCommand) && (s == mainForm)){
            destroyApp(false);
         }

      }

      public class runThread extends Thread{
         public void run(){
            if((mc == unregCommand) && (ms == mainForm)){
               try{
                  registeredConns = PushRegistry.listConnections(false);
                  if(unregForm.size() > 0) unregForm.delete(0);
                  registeredConnsCG = new ChoiceGroup("Connections", Choice-
Group.MULTIPLE, registeredConns, null);
                  if(registeredConnsCG.size() > 0) unreg-
Form.append(registeredConnsCG);
                  else unregForm.append("No registered connections found.");
                  display.setCurrent(unregForm);
               } catch (Exception e) {
                  showMessage("Unexpected " + e.toString() + ": " +
e.getMessage());
               }

            }
```

```
            if((mc == regCommand) && (ms == regForm)){
               try{
                  PushRegistry.registerConnection("sms://:" + regConnec-
tion.getString(), "Receive", regFilter.getString());
                  showMessage("Connection successfully registered");
               } catch (Exception e){
                  showMessage("Unexpected " + e.toString() + ": " +
e.getMessage());
               }
            }

            if((mc == unregCommand) && (ms == unregForm)){
               try{
                  if(registeredConnsCG.size() > 0){
                     for(int i=0; i<registeredConnsCG.size(); i++){
                        if(registeredConnsCG.isSelected(i)){
                           PushRegistry.unregisterConnection(registeredConnsCG.
getString(i));
                           registeredConnsCG.delete(i);
                           if(registeredConnsCG.size() == 0){
                              unregForm.delete(0);
                              unregForm.append("No registered connections found.");
                           }
                        }
                     }
                  }
               } catch (Exception e) {
                  showMessage("Unexpected " + e.toString() + ": " +
e.getMessage());
               }
            }
         }
           }
}
```

WakeUp.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.PushRegistry;
import javax.microedition.rms.*;
import java.util.*;
import javax.microedition.io.*;
```

```
public class WakeUp extends MIDlet implements CommandListener{

    public static Display display;
    public static Form mainForm;
    public static Command exitCommand;
    public static TextField tf;
    public static Command registerCommand;


    public void startApp() {

        display = Display.getDisplay(this);

        mainForm = new Form("WakeUp");
        exitCommand = new Command("Exit", Command.EXIT, 0);
        registerCommand = new Command("Register", Command.SCREEN, 0);
        tf = new TextField("Delay in seconds", "10", 10, TextField.NUMERIC);
        mainForm.addCommand(exitCommand);
        mainForm.addCommand(registerCommand);
        mainForm.append(tf);
        mainForm.setCommandListener(this);



        display.setCurrent(mainForm);

    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
        notifyDestroyed();
    }

    public void commandAction(Command c, Displayable s) {
        if((c == exitCommand) && (s == mainForm)){
            destroyApp(false);
        }
        if(c == registerCommand){

            new regThread().start();

        }
    }
```

```
   public class regThread extends Thread{


      public void run(){

         try {
            long delay = Integer.parseInt(tf.getString()) * 1000;

            long curTime = (new Date()).getTime();

            System.out.println(curTime + delay);

            PushRegistry.registerAlarm("WakeUp", curTime + delay);
            mainForm.append("Alarm registered successfully");

         } catch (NumberFormatException nfe) {
            mainForm.append("FAILED\nCan not decode delay " + nfe);
         } catch (ClassNotFoundException cnfe) {
            mainForm.append("FAILED\nregisterAlarm thrown " + cnfe);
         } catch (ConnectionNotFoundException cnfe) {
            mainForm.append("FAILED\nregisterAlarm thrown " + cnfe);
         }

      }
   }
}
```

SMS_send.java

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.PushRegistry;
import javax.wireless.messaging.*;
import javax.microedition.io.*;

public class SMS_send extends MIDlet implements CommandListener{

      public Display display;

      public static Form messageForm;
      public static Form mainForm;

      public static Command exitCommand;
```

```
        public static Command backCommand;
        public static Command sendCommand;

        public static TextField address_tf;
        public static TextField port_tf;
        public static TextField message_text_tf;

        String[] binary_str = {"Send BINARY message"};
        public static ChoiceGroup binary_cg;

        byte[] binary_data = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        String address;
        String text;

        MessageConnection conn = null;
        TextMessage txt_message = null;
        BinaryMessage bin_message = null;

        public SMS_send(){
            address_tf = new TextField("Address:", "", 32, TextField.PHONENUMBER);
            port_tf = new TextField("Port:", "1000", 32, TextField.PHONENUMBER);

            message_text_tf = new TextField("Message text:", "test message", 160,
TextField.ANY);
            binary_cg = new ChoiceGroup(null, Choice.MULTIPLE, binary_str, null);


            display = Display.getDisplay(this);

            messageForm = new Form("SMS_send");
            mainForm = new Form("SMS_send");

            exitCommand = new Command("Exit", Command.EXIT, 0);
            backCommand = new Command("Back", Command.BACK, 0);
            sendCommand = new Command("Send", Command.ITEM, 1);

            mainForm.append(address_tf);
            mainForm.append(port_tf);
            mainForm.append(message_text_tf);
            mainForm.append(binary_cg);
            mainForm.addCommand(exitCommand);
            mainForm.addCommand(sendCommand);
            mainForm.setCommandListener(this);

            messageForm.addCommand(backCommand);
```

```
        messageForm.setCommandListener(this);


    }

    public void pauseApp(){
    }

    protected void startApp() {
        display.setCurrent(mainForm);
    }

    public void destroyApp(boolean unconditional) {
        notifyDestroyed();
    }

    public void showMessage(String s) {
        if(messageForm.size() != 0 ) messageForm.delete(0);
        messageForm.append(s);
        display.setCurrent(messageForm);
    }

    public void commandAction(Command c, Displayable s) {
        if((c == backCommand) && (s == messageForm)){
            display.setCurrent(mainForm);
        }
        if((c == exitCommand) && (s == mainForm)){
            destroyApp(false);
        }
        if((c == sendCommand) && (s == mainForm)){
            address = "sms://" + address_tf.getString();
            if(port_tf.size() != 0) address += ":" + port_tf.getString();
            text = message_text_tf.getString();
            new send_thread().start();
        }
    }

    public class send_thread extends Thread{
        public void run(){
            try{
                conn = (MessageConnection) Connector.open(address);
                if(!binary_cg.isSelected(0)){
                    txt_message = (TextMessage)
conn.newMessage(MessageConnection.TEXT_MESSAGE);
                    txt_message.setPayloadText(text);
```

```
                conn.send(txt_message);
            } else {
                bin_message = (BinaryMessage)
conn.newMessage(MessageConnection.BINARY_MESSAGE);
                bin_message.setPayloadData(binary_data);
                conn.send(bin_message);
            }
            conn.close();
            showMessage("Message sent");
        } catch (Throwable t) {
            showMessage("Unexpected " + t.toString() + ": " + t.getMessage());
        }
    }
  }
}
```

Code Sample 10 Push Registry

22.9 Delivery of a Push Message

A push message intended for a MIDlet on the MOTORAZR V3xx handset will handle the following interactions:

MIDlet running while receiving a push message - if the application receiving the push message is currently running, the application will consume the push message without user notification.

No MIDlet suites running - if no MIDlets are running, the user will be notified of the incoming push message and will be given the option to run the intended application as shown in Figure 19 .

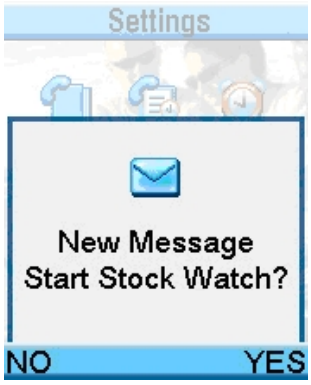


Figure 19 Intend Application Run Option

Push registry with Alarm/Wake-up time for application - push registry supports one outstanding wake-up time per MIDlet in the current suite. An application will use the TimerTask notification of time-based events while the application is running.

Another MIDlet suite is running during an incoming push - if another MIDlet is running, the user will be presented with an option to launch the application that had registered for the push message. If the user selects the launch, the current MIDlet is terminated.

Stacked push messages - it is possible for the handset to receive multiple push messages at one time while the user is running a MIDlet. The user will be given the option to allow the MIDlets to end and new MIDlets to begin. The user will be given the ability to read the messages in a stacked manner (stack of 5 supported), and if not read, the messages should be discarded.

No applications registered for push - if there are no applications registered to handle this event, the incoming push message will be ignored.

22.10 Deleting an Application Registered for Push

If an application registered in the Push Registry is deleted, the corresponding push entry will be deleted, making the PORT number available for future Push Registrations.

22.11 Security for Push Registry

Push Registry is protected by the security framework. The MIDlet registered for the push should have the necessary permissions. Details on permissions are outlined in the Security chapter.

22.12 Network Access

Untrusted applications will use the normal HttpConnection and HttpsConnection APIs to access web and secure web services. There are no restrictions on web server port numbers through these interfaces. The implementations augment the protocol so that web servers can identify untrusted applications. The following will be implemented:

- The implementation of HttpConnection and HttpsConnection will include a separate User-Agent header with the Product-Token "UNTRUSTED/1.0".User-Agent headers supplied by the application will not be deleted.
- The implementation of SocketConnection using TCP sockets will throw java.lang.SecurityException when an untrusted MIDlet suite attempts to connect on ports 80 and 8080 (http) and 443 (https).
- The implementation of SecureConnection using TCP sockets will throw java.lang.SecurityException when an untrusted MIDlet suites attempts to connect on port 443 (https).
- The implementation of the method DatagramConnection.send will throw java.lang.SecurityException when an untrusted MIDlet suite attempts to send datagrams to any of the ports 9200-9203 (WAP Gateway).
- The above requirements should be applied regardless of the API used to access the network. For example, the javax.microedition.io.Connector.open and javax.microedition.media.Manager.createPlayer methods should throw java.lang.SecurityException if access is attempted to these port numbers through a means other than the normal HttpConnection and HttpsConnection APIs.

23 Platform Request API

23.1 Platform Request API

The Platform Request API MIDlet package defines MIDP applications and the interactions between the application and the environment in which the application runs.

Refer to Table 30 for Platform Request API feature/class support for MIDP 2.0:

Feature/Class	Implementation
All constructors, methods, and inherited classes for the MIDlet class	Supported
platformRequest() method in javax.microedition.midlet	Supported
Will not support the "text/vnd.sun.j2me.app-descriptor" mime type in the URL for the platformRequest() support	Supported
Will not support the "application/java-archive" mime type in the URL for the platformRequest() method	Supported
Launching native apps with URLs	Supported
URL compatible launch of the WAP Browser	Supported
URL compatible launch of the phone dialer	Supported
Will not require the MIDlet to exit in order to launch an application from the platformRequest() method	Supported
Will pause the MIDlet when executing the platformRequest() method.	Supported
Will resume the MIDlet after the user exits the application launched by the platform Request() method.	Supported, resumes to Java Service Menu
All constructors and inherited methods for the MIDletStateChangeException in javax.microedition.midlet	Supported

Table 30 Platform Request API feature/class support for MIDP

For MIDP 2.0, the javax.microedition.midlet.MIDlet.platformRequest () method

should be used and called when the MIDlet is destroyed. The following code sample is an example of the Platform Request API:

Start a Call

```
MIDlet.platformrequest("tel:88143593")
```

Start a Web Session

```
MIDlet.platformrequest("http://gonzaga.cesar.org.br/
~bam/triplets/tii/menu.wml")

MIDlet.platformrequest("http://gonzaga.cesar.org.br/
~bam/triplets/tii/Millionaire1.jad");
```

Code Sample 11 Plataform Request

23.2 MIDlet Request of a URL that Interacts with Browser

When a MIDlet suite requests a URL, the browser will come to the foreground and connect to that URL. The user will then have access to the browser and control over any downloads or network connections. The initiating MIDlet suite will continue running in the background, if it cannot (upon exiting the requesting MIDlet suite) the handset will bring the browser to the foreground with the specified URL.

If the URL specified refers to a MIDlet suite, JAD, or JAR, the request will be treated as a request to install the named package. The user will be able to control the download and installation process, including cancellation. Please note normal Java installation process should be used.

Refer to the JAD Attributes chapter for more details.

23.3 MIDlet Request of a URL that Initiates a Voice Call

If the requested URL takes the form `tel: <number>`, the handset will use this request to initiate a voice call as specified in RFC2806. If the MIDlet will be exited to handle the URL request, the handset will only handle the last request made. If the MIDlet suite continues to run in the background when the URL request is being made, all other requests will be handled in a timely manner.

The user will be asked to acknowledge each request before any actions are taken by the handset, and upon completion of the platform request, the Java Service Menu will be displayed to the user.

24 JAD Attributes

24.1 JAD / Manifest Attribute Implementations

The JAR manifest defines attributes to be used by the application management software (AMS) to identify and install the MIDlet suite. These attributes may or may not be found in the application descriptor.

The application descriptor is used, in conjunction with the JAR manifest, by the application management software to manage the MIDlet. The application descriptor is also used for the following:

- By the MIDlet for configuration specific attributes
- Allows the application management software on the handset to verify the MIDlet is suited to the handset before loading the JAR file
- Allows configuration-specific attributes (parameters) to be supplied to the MIDlet(s) without modifying the JAR file.

Motorola has implemented the following support for the MIDP 2.0 Java Application Descriptor attributes as outlined in the JSR-118. Table 31 lists all MIDlet attributes, descriptions, and its location in the JAD and/or JAR manifest that are supported in the Motorola implementation. Please note that the MIDlet will not install if the MIDlet-Data-Size is greater than 512k.

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlets to the user	Yes	Yes
MIDlet-Version	The version number of the MIDlet	Yes	Yes

	suite		
MIDlet-Vendor	The organization that provides the MIDlet suite.	Yes	Yes
MIDlet-Icon	The case-sensitive absolute name of a PNG file within the JAR used to represent the MIDlet suite.	Yes	Yes
MIDlet-Description	The description of the MIDlet suite.	No	No
MIDlet-Info-URL	A URL for information further describing the MIDlet suite.	Yes	No
MIDlet-<n>	The name, icon, and class of the nth MIDlet in the JAR file. Name is used to identify this MIDlet to the user. Icon is as stated above. Class is the name of the class extending the javax.microedition.midlet. MIDlet-class.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MIDlet-Jar-URL	The URL from which the JAR file can be loaded.		Yes
MIDlet-Jar-Size	The number of bytes in the JAR file.		Yes
MIDlet-Data-Size	The minimum number of bytes of persistent data required by the MIDlet.	Yes	Yes
MicroEdition-Profile	The Java ME profiles required. If any of the profiles are not implemented the installation will fail.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MicroEdition-Configuration	The Java ME Configuration required, i.e CLDC	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MIDlet-Permissions	Zero or more permissions that are	Yes	Yes

	critical to the function of the MIDlet suite.		
MIDlet-Permissions-Opt	Zero or more permissions that are non-critical to the function of the MIDlet suite.	Yes	Yes
MIDlet-Push-<n>	Register a MIDlet to handle inbound connections	Yes	Yes
MIDlet-Install-Notify	The URL to which a POST request is sent to report installation status of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Notify	The URL to which a POST request is sent to report deletion of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Confirm	A text message to be provided to the user when prompted to confirm deletion of the MIDlet suite.	Yes	Yes

Table 31 MIDlet attributes, descriptions, and its location in the JAD and/or JAR manifest

25 LCDUI

25.1 LCDUI API

Table 32 lists the specific interfaces supported by Motorola implementation:

Interface	Description
Choice	Choice defines an API for user interface components implementing selection from a predefined number of choices.
CommandListener	This interface is used by applications which need to receive high-level events from implementation
ItemCommandListener	A listener type for receiving notification of commands that have been invoked on Item286 objects
ItemStateListener	his interface is used by applications which need to receive events that indicate changes in the internal state of the interactive items within a Form231 screen.

Table 32 LCDUI API specific interfaces supported by Motorola implementation

Table 33 lists the specific classes supported by Motorola implementation:

Classes	Description
Alert	An alert is a screen that shows data to the user and waits for a certain period of time before proceeding to the next Displayable.
AlertType	The AlertType provides an indication of the nature of alerts.
Canvas	The Canvas class is a base class for writing applications that need to handle low-level events and to issue graphics calls for drawing to the display.
ChoiceGroup	A ChoiceGroup is a group of selectable elements intended to be placed within a Form.
Command	The Command class is a construct that encapsulates the semantic information of an action.

CustomItem	A CustemItem is customizable by sub classing to introduce new visual and interactive elements into Forms.
DateField	A DateField is an editable component for presenting date and time (calendar) information that will be placed into a Form.
Display	Display represents the manager of the display and input devices of the system.
Displayable	An object that has the capability of being placed on the display.
Font	The Font class represents fonts and font metrics.
Form	A Form is a Screen that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, choice groups, and custom items.
Gauge	Implements a graphical display, such as a bar graph of an integer value.
Graphics	Provides simple 2D geometric rendering capability.
Image	The Image class is used to hold graphical image data.
ImageItem	An item that can contain an image.
Item	A superclass for components that can be added to a Form.
List	A Screen containing a list of choices.
Screen	The common superclass of all high-level user interface classes.
Spacer	A blank, non-interactive item that has a settable minimum size.
StringItem	An item that can contain a string.
TextBox	The TextBox class is a Screen that allows the user to enter and edit data.
TextField	A TextField is an editable text component that will be placed into a Form.
Ticker	Implements a "ticker-tape", a piece of text that runs continuously across the display.

Table 33 LCDUI API specific classes supported by Motorola implementation

Refer to Table 34 for LCDUI feature/class support for MIDP 2.0:

Feature/Class	Implementation
All fields, constructors, methods, and inherited methods for the Alert class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, and inherited methods for the AlertType class in the javax.microedition.lcdui package	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of ALARM	Supported

Will provide and play an audible sound when the play Sound() method is called with an AlertType of ERROR	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of WARNING	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of CONFIRMATION	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of INFO	Supported
All fields, constructors, methods, and inherited methods for the Canvas Class in the javax.microedition.lcdui. package	Supported
Status indicators out of full-screen mode will consume a portion of the display	Supported
UP field in javax.microedition.lcdui.Canvas to the top position of the key	Supported
DOWN field in javax.microedition.lcdui.Canvas to the bottom position of the key	Supported
LEFT field in javax.microedition.lcdui.Canvas to the left position of the key	Supported
RIGHT field in javax.microedition.lcdui.Canvas to the right position of the key	Supported
All fields and methods for the Choice interface in the javax.microedition.lcdui package	Supported
Truncate an image in a Choice object if it exceeds the capacity of the device display	Supported
Truncation of very long elements will not occur in a Choice object	Text in forms is wrapped and scrolled
Will display a portion of long elements to display and provide a means for the user to view all of the parts of the element	Supported
Truncation in elements w/line breaks will not occur in a Choice object	Supported
Portion of line break elements to display and provide a means for the user to view all parts of the element	Supported
All constructors, methods, inherited fields, and inherited methods for the ChoiceGroup class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the Command class in the javax.microedition.lcdui package	Supported
All methods for the CommandListener interface in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the CustomItem abstract class in the	Supported

javax.microedition.lcdui package	
All fields, constructors, methods, inherited fields, and inherited methods for the DateField class in the javax.microedition.lcdui package	Supported
All fields, methods, and inherited methods for the Display class in the javax.microedition.lcdui package	Supported
Maximum colors for the numColors() method in javax.microedition.lcdui.Display	64K colors supported
All methods and inherited methods for the Displayable class in the javax.microedition.lcdui package	Supported
Adding commands to soft buttons before placing it in a menu for the addCommand() method in javax.microedition.lcdui.Displayable	Supported
All fields, methods, and inherited methods for the Font class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the FORM class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the Gauge class in the javax.microedition.lcdui package	Supported
All fields, methods, and inherited methods for the Graphics class in the javax.microedition.lcdui package	Supported
DOTTED stroke style	Supported
SOLID stroke style	Supported
All methods and inherited methods for the Image class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the ImageItem class in the javax.microedition.lcdui package	Supported
All fields, methods, and inherited methods for the Item class in the javax.microedition.lcdui package	Supported
Label field	Supported
All methods for the ItemCommandListener interface in the javax.microedition.lcdui package	Supported
All methods ItemStateListener interface in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the List class in the javax.microedition.lcdui package	Supported
All constructors, methods, inherited fields, and inherited methods for the Spacer class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the StringItem class in the javax.microedition.lcdui package	Supported

All constructors, methods, and inherited methods for the TextBox class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the TextField class in the javax.microedition.lcdui package	Supported
Visual indication with UNEDITABLE field set will be provided	Supported
All constructors, methods, and inherited methods for the Ticker class in the javax.microedition.lcdui package	Supported
OEM Lights API providing control to the lights present on the handset	Supported, Fun Lights API
All fields, constructors, methods, inherited fields, and inherited methods for the TextField class in the havax.microedition.lcdui package	Supported

Table 34 Feature/class support for MIDP

26 iTAP

26.1 Intelligent Keypad Text Entry API

When users are using features such as SMS (short message service), or "Text Messaging", they can opt for a predictive text entry method from the handset. The Java ME environment has the ability to use SMS in its API listing. The use of a predictive entry method is a compelling feature to the MIDlet.

This API will enable a developer to access iTAP, Numeric, Symbol and Browse text entry methods. With previous Java ME products, the only method available was the standard use of TAP.

Predictive text entry allows a user to simply type in the letters of a word using only one key press per letter, as apposed to the TAP method that can require as many as four or more key presses. The use of the iTAP method can greatly decrease text-entry time. Its use extends beyond SMS text messaging, but into other functions such as phonebook entries.

The following Java ME text input components will support iTAP.

- javax.microedition.lcdui.TextBox

The TextBox class is a Screen that allows the user to edit and enter text.

- javax.microedition.lcdui.TextField

A TextField is an editable text component that will be placed into a Form. It is given a piece of text that is used as the initial value.

Refer to Table 35 for iTAP feature/class support for MIDP 2.0:

Feature/Class
Predictive text capability will be offered when the constraint is set to ANY
User will be able to change the text input method during the input process when the constraint is set to ANY (if predictive text is available)
Multi-tap input will be offered when the constraint on the text input is set to EMAIL-ADDR, PASSWORD, or URL

Table 35 iTAP feature/class

27 Java.lang Implementation

27.1 java.lang support

Motorola implementation for the `java.lang.System.getProperty` method will support additional system properties beyond what is outlined in the JSR-118 specification.

The additional system properties are as follows:

- Cell ID: The current Cell ID of the device will be returned during implementation.
- Battery Level: The current battery level of the application will be returned during implementation. Battery values are the following: low battery, 1, 2, and 3, based on the battery level.
- IMEI: The IMEI number of the device will be returned during implementation.
- MSISDN: The MSISDN of the device will be returned during implementation.

The IMEI and MSISDN properties will not be available for unsigned MIDlets. The Code Sample 12 shows the java.lang implementation:

```
System.getProperty("batterylevel")
System.getProperty("MSISDN")
System.getProperty("CellID")
System.getProperty("IMEI")
```

Code Sample 12 Java.lang implementation

28 CommConnection Interface

28.1 CommConnection

The CommConnection interface defines a logical serial port connection. A logical serial port connection is a logical connection through which bytes are transferred serially. This serial port is defined within the underlying operating system and may not correspond to a physical RS-232 serial port. For example, IrDA IRCOMM ports can be configured as a logical serial port within the operating system so it can act as a logical serial port.

28.2 Accessing

The Comm port is accessed using a Generic Connection Framework string with an explicit port identifier and embedded configuration parameters, each separated with a semi-colon (;). Only one application may be connected to a particular serial port at a given time. A `java.io.IOException` is thrown if an attempt is made to open the serial port with `Connector.open()` if the connection is already open.

A URI with the type and parameters is used to open the connection. The scheme, as defined in RFC 2396, will be the following:

- `Comm.: <port identifier> [<optional parameters>]`

28.3 Parameters

The first parameter will be a port identifier, which is a logical device name. These port identifiers are The valid identifiers for a particular device and OS can be queried through the `System.getproperty()` method using the key microedition.commports. A list of ports, separated by commas, is returned which can be combined with a comm: prefix as the URL string to open a serial port connection.device specific and should be used with care.

The valid identifiers for a particular device and OS can be queried through the `System.getproperty()` method using the key `microedition.commports`. A list of ports, separated by commas, is returned which can be combined with a comm: prefix as the URL string to open a serial port connection.

Any additional parameters will be separated by a semi-colon (;) without spaces. If a particular parameter is not applicable to a particular port, the parameter will be ignored. The port identifier cannot contain a semi-colon (;).

Legal parameters are defined by the definition of the parameters below. Illegal or unrecognized parameters cause an `IllegalArgumentException`. If the value of a parameter is supported by the device, it will be honored. If the value of a parameter is not supported, a `java.io.IOException` is thrown. If a baudrate parameter is requested, it is treated the same way that a `setBaudRate` method handles baudrates. For example, if the baudrate requested is not supported, the system will substitute a valid baudrate which can be discovered using the `getBaudRate` method.

Table 36 describes optional parameters.

Parameter	Default	Description
baudrate	platform dependent	The speed of the port.
bitsperchar	8	The number bits per character(7 or 8).
stopbits	1	The number of stop bits per char(1 or 2)
parity	none	The parity can be odd, even, or none.
blocking	on	If on, wait for a full buffer when reading.
autocts	on	If on, wait for the CTS line to be on before writing.
autorts	on	If on, turn on the RTS line when the input buffer is not full. If off, the RTS line is always

		on.

Table 36 Interface Commconncetion optional parameters

28.4 BNF Format for Connector.open () string

The URI must conform to the BNF syntax specified in Table 37. If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

BNF syntax	
<comm_connection_string>	::= "**comm:**"<port_id>[<options_list>] ;
<port_id>	::= *string of alphanumeric characters*
<options_list>	::= *(<baud_rate_string>| <bitsperchar>| <stopbits>| <parity>| <blocking>| <autocts>| <autorts>) ; ; if an option duplicates a previous option in the ; option list, that option overrides the previous ; option
<baud_rate_string>	::= ";**baudrate**="<vbaud_rate>
<baud_rate>	::= *string of digits*
<bitsperchar>	::= ";**bitsperchar**="<bit_value>
<bit_value>	::= "7" | "8"
<stopbits>	::= ";**stopbits**="<stop_value>
<stop_value>	::= "1" | "2"
<parity>	::= ";**parity**="<parity_value>
<parity_value>	::= "even" | "odd" | "none"
<blocking>	::= ";**blocking**="<on_off>
<autocts>	::= ";**autocts**="<on_off>
<autorts>	::= ";**autorts**="<on_off>
<on_off>	::= "on" | "off"

Table 37 Interface Commconncetion BNF syntax

28.5 Comm Security

Access to serial ports is restricted to prevent unauthorized transmission or reception of data. The security model applied to the serial port connection is defined in the im-

plementing profile. The security model will be applied on the invocation of the `Connector.open ()` method with a valid serial port connection string. Should the application not be granted access to the serial port through the profile authorization scheme, a `java.lang.SecurityException` will be thrown from the `Connector.open ()` method. The security model will be applied during execution, specifically when the methods `openInputStream(), openDataInputStream(), openOutputStream(), and openDataOutputStream()` are invoked.

Code Sample 13 shows the implementation of CommConnection:

Sample of a CommConnection accessing a simple loopback program

```
CommConnection cc = (CommConnection)
          Connector.open("comm:com0;baudrate=19200");
 int baudrate = cc.getBaudRate();
 InputStream is  = cc.openInputStream();
 OutputStream os = cc.openOutputStream();
 int ch = 0;
 while(ch != 'Z') {
    os.write(ch);
    ch = is.read();
    ch++;
 }
 is.close();
 os.close();
 cc.close();
```

Sample of a CommConnection discovering available comm Ports

```
String port1;
 String ports = System.getProperty("microedition.commports");
 int comma = ports.indexOf(',');
 if (comma > 0) {
    // Parse the first port from the available ports list.
    port1 = ports.substring(0, comma);
 } else {
    // Only one serial port available.
    port1 =ports;
 }
```

Code Sample 13 CommConnection implementation

28.6 Port Naming Convention

Logical port names can be defined to match platform naming conventions using any combination of alphanumeric characters. Ports will be named consistently among the implementations of this class according to a proposed convention. VM implementations will follow the following convention:

- Port names contain a text abbreviation indicating port capabilities followed by a sequential number for the port. The following device name types will be used:
  - COM# - COM is for RS-232 ports and # is a number assigned to the port
  - IR# - IR is for IrDA IRCOMM ports and # is a number assigned to the port

The naming scheme allows API users to determine the type of port to use. For example, if an application "beams" a piece of data, the application will look for IR# ports for opening the connection.

28.7 Method Summary

Table 38 describe the CommConnection method summary for MIDP 2.0.

Method Summary	
int	`getBaudRate()` Gets the baudrate for the serial port connection
int	`setBaudRate (int baudrate)` Sets the baudrate for the serial port connection

Table 38 Method Summary

29 Motorola Get URL from Flex API

29.1 Overview

This feature allows accessing URL stored in FLEX by a Java application. Carriers flex the URL, which is used for content download, into the phone just like any invisible net URL. So, this feature would allow Java applications to read and display the URL stored in flex for users that would like to download new levels of Game.

The existing functionality allows current Java Applications use a dedicated URL to inform users about the location which a new level of game can be downloaded. This new functionality allows carriers to specify the URL for content download.

29.2 Flexible URL for downloading functionality

The URL is flexed using RadioComm or using OTA provisioning. The URL will follow the rules mentioned below:

- All URLs used shall follow the guidelines outlined in RFC1738: Uniform Resource Locators (URL). Refer to *http://www.w3.org/addressing/rfc1738.txt* for more information.
- URLs are limited to 128 characters.

This feature enables Java applications to read the URL stored at the predefined loca-

tion in flex table.

The Java Application will be able to access the flexed URL by System.getProperty method. The key for accessing the URL is "com.mot.carrier.URL". The method System.getProperty will return NULL if no URL is flexed.

29.3 Security Policy

Only trusted applications will be granted permission to access this property.

30 Motorola Secondary Display API

This chapter details the capability for Java ME applications to render content to Motorola devices that feature a secondary display.

Motorola P2K Java ME ™ enabled devices currently support the standard low-level and high-level APIs for UI development for the primary display only. When the focus is removed from the kJava environment by closing the flip (or exiting the environment) the Java ME ™ application cannot display any text or graphical information to the secondary display.

Motorola devices that feature a secondary display will provide the capability to extend application UI to the secondary display.

30.1 Primary Requirements

The first implementation of the Secondary Display API is targeted for the Triplets refresh. Supporting Displayable items such as Forms, TextBox, List is not required, although design considerations will be made to support these UI features in future releases.

The Secondary Display API will provide functionality to access the secondary display:

- The secondary display API must not support the Displayable sub-class of Screen or Screen's sub-classes (Form, TextBox, etc.). Screen and its subclasses support high-level layout and input support which is not needed in this version of Secondary Display API.

- The secondary display API must not support any input elements like Choice, Item, Text-Field etc.
- Secondary display API must support setting Ticker on secondary display.
- The Secondary Display API must support key event processing. Key mappings will be supported for Voice and Camera/Smart keys. Extra keys shall be supported depending on device requirements.
- Only one display, either primary or secondary shall be having focus at a given time. Primary display shall be active when flip is open and secondary display shall be active when flip is closed. Events including key events shall be delivered to the current active display only.
- The secondary canvas must support full screen and normal modes. In full screen mode, the whole secondary display area will be available for the MIDlet. In normal mode, the status area won't be available for display.
- The secondary display API must support all Graphics class functionality.
- Providing some game canvas functionality to the secondary display is not required first implementation, but will be taken up for future releases.
- Multimedia resources must be available for MIDlets running on secondary display playing audio media and decoding images when the flip is closed.
- Secondary Display API must support Flip open, Flip close event processing.

30.2 Flip-Open, Flip-Closed Event Handling

A running MIDlet can continue to run on the secondary display when the flip is closed.

A MIDlet running on secondary display can switch to primary display if the flip is opened.

The MIDlet shall receive flip open, flip close events and can take appropriate action based on these events.

30.3 Exception Handling

For portability purposes, the design of the API will allow the developer to handle Exceptions related to the instantiation of the secondary display context. Appropriate exceptions should be generated for invocation of methods not supported by secondary display.

30.4 Push enabled applications

While the flip is closed, it is desirable to start up MIDlets if a push is received on a registered port and the associated MIDlet can run on secondary display. This will be subject to user confirmation.

This feature need not be implemented for the first version of API.

30.5 Feature interaction

Any incoming call, message or any scheduled native application should have priority over MIDlet running on secondary display. If any native application requests focus, the running MIDlet shall be suspended.

30.6 Low power requirements

It is desirable that the VM runs on low-power mode while using the secondary display.

30.7 Security

Secondary Display API shall follow MIDP 2.0 security model.

Appendix A
Key Mapping

Key Mapping

Table 39 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 39 Key Mapping

Appendix B Memory Management Calculation

The available memory on the MOTORAZR V3xx is the following:

- 64 MB shared memory for MIDlet storage
- 4Mb Heap size

Appendix C
FAQ

The MOTODEV developer program is online and provides access to Frequently Asked Questions about enabling technologies on Motorola products.

Access to dynamic content based on questions from the Motorola Java ME developer community is available at the URL stated below.

http://developer.motorola.com/

Appendix D
HTTP Range

Graphic Description

Figure 20 shows a graphic description of HTTP Range:

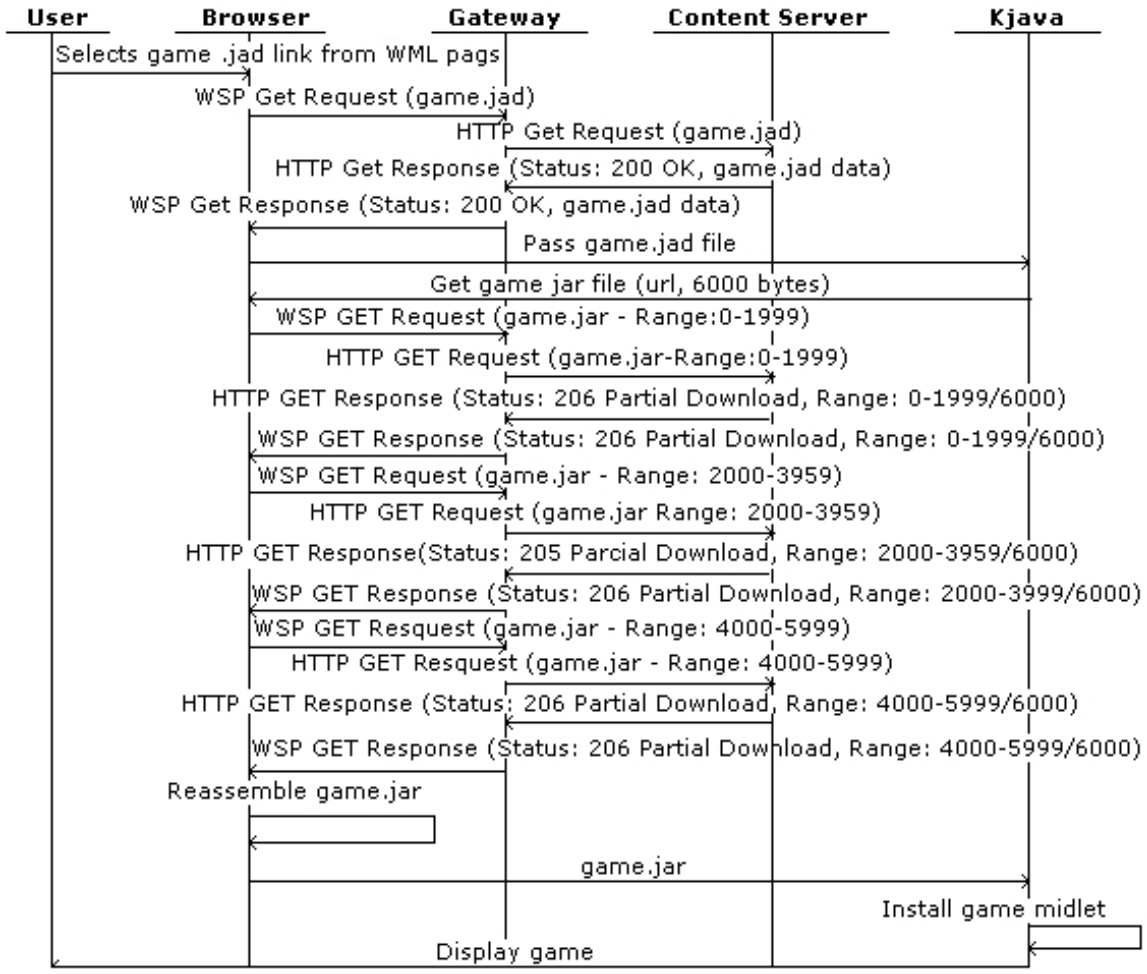


Figure 20 Graphic Description of HTTP Range

Appendix F
Spec Sheet

Spec Sheet

Listed below is the spec sheets for the MOTORAZR V3xx. The spec sheet contains information regarding the following areas:

- Technical Specifications
- Key Features
- Java ME Information
- Motorola Developer Information
- Tools
- Other Related Information

MOTORAZR V3xx
Developer Reference Sheet

Technical Specifications

Band/Frequency	E-GSM900/GSM 1800/GSM 1900/GPRS/EDGE/HSDPA 3.6mbps
Region	EMEA/Asia/Australia
Technology	UMTS, GSM, EDGE, GPRS
Connectivity	Mini-USB v2.0, Bluetooth
Dimensions	53 x 104.5 x 15.5 mm
Weight	105g
Display	96 x 80 (65k colors) 240 x 320 (262k colors)
Operating System	Motorola

Key Features

- Updated & streamlined MOTORAZR V3xx design
- HSDPA 3.6 Mbps (category 6)
- EDGE Class 10, GPRS Class 10
- Integrated VGA and 1.3 megapixel digital camera with 8x digital zoom and full screen viewfinder
- Integrated MP3 player with Media Finder and optional mini-USB stereo headset
- Optional, hot swappable microSD memory card
- MPEG4 video capture and playback
- Integrated Class 2 Bluetooth wireless technology
- Bluetooth stereo Music Profile (A2DP) and Bluetooth Music Control Profile (AVRCP) for streaming music to compatible Bluetooth enabled wireless stereo headphones
- VGA Imager Point to Point Video at up to 15 frames per second

Java ME Information

CLDC v1.1 and MIDP v2.0 compliant	
Heap Size	4Mb
Maximum record store size (RMS)	512 KB
MIDlet storage avaliable	64 MB
Interface connections	HTTP, HTTPS, Datagram, Socket Stream, Secure Socket(Stream/Layer)
Maximum number of Sockets	4
Image Support	.jpeg .gif .png .bmp
Double Buffering	Supported
Encoding schemes	ISO-8859-1, UTF-8/16
Input methods	iTap/Multitap
Additional APIs	JSR-75, JSR-82, JSR-118, JSR-120, JSR-135, JSR-139, JSR-177, JSR-184, JSR-185, JSR-205, Motorola Get URL from Flex API, Motorola Secundary Display API
Audio Support	AAC, AAC+, Enh AAC+, AMR NB/WB,

Related Information

Motorola Developer Information:
Developer Resources at
http://developer.motorola.com/
Tools:
Motorola Java™ ME SDK version v6.1 SE
Motorola Messaging Suite v1.1
Documentation:
Creating Media for the MOTORAZR V3xx Handset
Purchase:
Visit the MOTODEV Shop at http://developer.motorola.com/
Accessories: http://www.motorola.com/consumer

References:
Java ME specifications:
http://java.sun.com/javame/
MIDP v2.0 specifications:
http://www.java.sun.com/products/midp
CLDC v1.0/v1.1 specifications:
http://www.java.sun.com/products/cldc
WAP forum: http://www.wapforum.org
EMS standards: http://www.3GPP.org

Updated on 18-Dec-2006.

Appendix H
Quick Reference