Technical Manual

Motorola C650 Handset

J2ME™ Developer Guide

Version 1.4

Table of Contents

1 Introduction

Purpose

This document describes the application program interfaces used to develop Motorola compliant Java™ 2 Platform, Micro Edition (J2ME™) applications for the C650 handset.

Audience

This document is intended for premium J2ME developers and specific carriers involved with the development of J2ME applications for the C650 handset.

Disclaimer

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

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

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

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

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

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

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

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

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

Motorola recommends that if you are not the sole 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.

References

Reference	Link
RFC 2068	http://ietf.org/rfc/rfc2068.txt
SAR	http://www.wapforum.org
MIDP 2.0	http://java.sun.com/products/midp/
JSR 118	http://www.jcp.org
JSR 120	http://www.jcp.org
JSR 135	http://www.jcp.org
Sun MIDP 2.0 SDK	http://java.sun.com/products/midp/
TLS protocol version 1.0	http://www.ietf.org/rfc/rfc2246.txt
SSL protocol version 3.0	http://home.netscape.com/eng/ssl3/draft302.txt
GSM 03.38 standard	http://www.etsi.org

GSM 03.40 standard	http://www.etsi.org
RFC 2437	http://ietf.org/rfc/rfc2437.txt
Sun J2ME	http://java.sun.com/j2me/).

Revision History

Version	Date	Name	Reason
0.1	December 17, 2003	Motorola	Initial Draft
1.0	January 7, 2003	Motorola	Baseline
1.4	July 25, 2004	Motorola	Updated Version

Definitions, Abbreviations, Acronyms

Acronym	**Description**
AMS	Application Management Software
API	Application Program Interface.
CLDC	Connected Limited Device Configuration
GPS	Global Positioning System
IDE	Integrated Development Environment
ITU	International Telecommunication Union
JAD	Java Application Descriptor
JAL	Java Application Loader
JAR	Java Archive. Used by J2ME applications for compression and packaging.
J2ME	Java 2 Micro Edition
JSR 120	Java Specification Request 120 defines a set of optional APIs that provides standard access to wireless communication resources.
JVM	Java Virtual Machine
KVM	Kilo Virtual Machine
MIDP	Mobile Information Device Profile

MMA	Multimedia API
MT	Mobile Terminated
OEM	Original Equipment Manufacturer
OTA	Over The Air
RMS	Record Management System
RTOS	Real Time Operating System
SDK	Software Development Kit
SMS	Short Message Service
SMSC	Short Messaging Service Center
SU	Subscribe Unit
UI	User Interface
URI	Unified Resource Identifier
VM	Virtual Machine
WMA	Wireless Messaging API

Document Overview

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

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

Chapter 2 – J2ME Introduction: this chapter describes the J2ME platform and the available resources on the Motorola C650 handset.

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

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

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

Chapter 6 -- Background Applications: this chapter describes all background applications.

Chapter 7 – Record Management System: this section describes the Record Management System API.

Chapter 8 – JAD Attributes: this chapter describes what attributes are supported.

Chapter 9 – LCDUI: this chapter describes the Limited Connected Device User Interface API.

Chapter 10 – Gaming API/Multiple Key Press: this chapter describes the Gaming API.

Chapter 11 -- Vibe and Backlight API – this chapter describes the Vibe and Backlight API.

Chapter 12 -- Fun Lights API – this chapter describes the Fun Lights API functionality.

Chapter 13 – Java.lang Implementation: this chapter describes the java.lang implementation.

Chapter 14 – iTAP: this chapter describes iTAP support.

Chapter 15 – Networking APIs: this chapter describes the Java Networking API.

Chapter 16 – CommConnection Interface: this chapter describes the CommConnection API.

Chapter 17 – Platform Request API: this chapter describes the platform request APIs.

Chapter 18 – JSR 135 Mobile Media: this chapter describes image types and supported formats.

Chapter 19 – JSR 120 Wireless Messaging API: this chapter describes JSR 120 implementation.

Chapter 20 -- Phonebook Access: this chapter describes the Phonebook Access API.

Chapter 21 – File System Access: this chapter describes the File System Access API.

Chapter 22 – MIDP 2.0 Security Model: this chapter describes the MIDP 2.0 default security model.

Appendix A – Key Mapping: this appendix describes the key mapping of the Motorola C650, including the key name, key code, and game action of all Motorola keys.

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

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

Appendix D – HTTP Range: this appendix provides a graphic description of HTTP Range.

Appendix E – Spec Sheet: this appendix provides the spec sheet for the Motorola C650 handset.

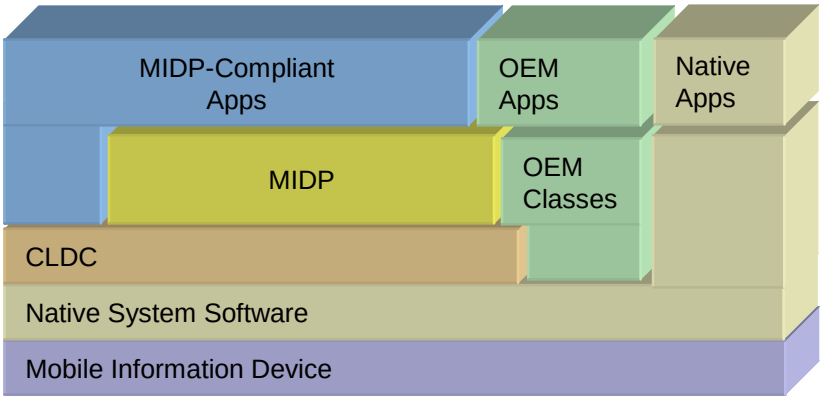
2
J2ME Introduction

The Motorola C650 handset includes the Java™ 2 Platform, Micro Edition, also known as the J2ME platform. The J2ME platform enables developers to easily create a variety of Java applications ranging from business applications to games. Prior to its inclusion, services or applications residing on small consumer devices like cell phones could not be upgraded or added to without significant effort. By implementing the J2ME platform on devices like the Motorola C650 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 J2ME environment and the tools that can be used to develop applications for the Motorola C650 handset.

The Java 2 Platform, Micro Edition (J2ME)

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

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



The Configuration Layer used in the Motorola C650 handset is the Connected Limited Device Configuration 1.0 (CLDC 1.0) and the Profile Layer used is the Mobile Information Device Profile 2.0 (MIDP 2.0). Together, the CLDC and MIDP provide common APIs for I/O, simple math functionality, UI, and more.

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

The Motorola J2ME Platform

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

The Motorola C650 handset contains OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the Motorola C650 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.

Resources and API's Available

MIDP 2.0 will provide support to the following functional areas on the Motorola C650 handset:

MIDP 2.0

- Application delivery and billing
- Application lifecycle
- Application signing model and privileged security model
- End-to-end transactional security (HTTPS)
- MIDlet push registration (server push model)
- Networking
- Persistent storage
- Sounds
- Timers
- User Interface
- File Image Support (.PNG, .JPEG, .GIF)

Additional Functionality

- WMA (JSR 120)
- MMA (JSR 135)
- Fun Lights API
- Phonebook API
- File System API

3
Developing and Packaging J2ME Applications

Guide to Development in J2ME

Introduction to Development

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

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

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

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

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

- For MIDP 1.0; JSR 37 should be reviewed.
- For MIDP 2.0; JSR 118 should be reviewed.
- For CLDC 1.0.4; JSR 30 should be reviewed.
- For CLDC 1.1; JSR 139 should be reviewed.

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

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

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

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

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

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

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

4
Downloading Applications

Methods of Downloading

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

Method 1 - OTA

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

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

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

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

During a download of the application, the user will see the MIB 2.2 browser displaying 'Downloading' followed by "x% completed" for either SAR or HTTP Byte Range type downloads.

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

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

The following error codes are supported:

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

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

Possible Screen Messages Seen With Downloading:

- If JAR -file size does not match with specified size, it displays “Failed Invalid File”. Upon time-out, the handset goes back to browser.
- When downloading is done, the handset displays a transient notice “Download Completed” and starts to install the application.
- Upon completing installation, the handset displays a transient notice “Installed” and returns to Browser after time-out.
- If the MANIFEST file is wrong, the handset displays a transient notice “Failed File Corrupt” and returns to Browser after time-out.

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

Method 2 - Direct Cable & Motorola MIDway Tool

The direct cable approach can be performed using a tool available from MOTOCODER called MIDway. The version available of writing is 2.6, which supports USB cable download. In order to use the tool properly, review FAQ 64 which contains the following about downloading:

1. MIDway 2.7.x executable
2. USB Driver for the cable to handset
3. Instructions on installation
4. User Guide for 2.7.x MIDway

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

- USB Cable Part Number (SKN6311A).

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 on the phone in question and seeing if a “Java app loader” option is available on that menu. If it is not, then contact MOTOCODER support for advice on how to receive an enabled handset.

Motorola provides a User Guide with the MIDway tool (as listed above) as well as a document outlining the tool for version 2.6 on the MOTOCODER website entitled “Installing J2ME MIDlet using MIDway Tool”.

The USER-AGENT String

The following table describes USER_AGENT strings associated with Motorola devices:

Motorola Device	USER_AGENT STRING
C650	`User-Agent: MOT-C650/xx.xx.xxR MIB/2.2 Profile/MIDP-2.0 Configuration/CLDC-1.0`

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

This table above provides information about the following components:

1. WAP Browser Release, Motorola Internet Browser (MIB) 2.2
2. MIDP version 2.0
3. CLDC version 1.0
4. The Phone Software Version is detailed by xx.xx.xx

Error Logs

The following table represents the error logs associated with downloading applications.

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

		Checksum of JAR file is not equal to Checksum in MIDlet-JAR-SHA attribute Application not authorized
Application Expired	MIDlet Launching	Security Certificates expired or removed
Application Error	MIDlet Execution	Authorization failure during MIDlet execution Incorrect MIDlet

5 Application Management

The following sections describe the application management scheme for the Motorola C650 handset. This chapter will discuss the following:

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

Downloading a JAR file without a JAD

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

MIDlet Upgrade

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

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

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

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

Installation and Deletion Status Reports

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

Refer to the table below for application management feature/class support for MIDP 2.0:

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

6 Background Applications

Background Attribute

A Motorola specific JAD attribute called background exists. MIDlets with JAD file containing Background = True can run in the background mode.

Background Java Application Lifecycle

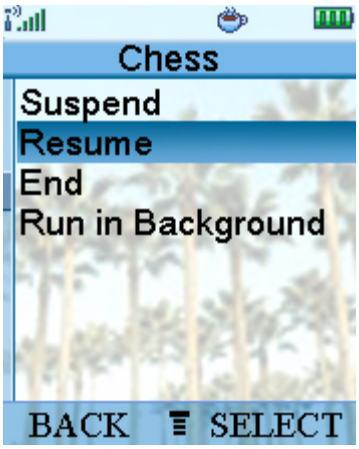
A MIDlet with background attributes will continue running when not in focus (in the background mode). MIDlets are able to accept incoming data if they are running in the background.

For example:

- The phonebook application can automatically synchronize new entries when in background mode.

Background MIDlet

When a MIDlet with background attributes is running, the user can press the END (red) key to initiate the following options shown below:



Pressing the END key will force the handset to display a Java service menu with the above options listed.

If the user selects to run the application in the background, the MIDlet will run in the background without focus. A Java icon will be displayed in the status bar to indicate to the user that a MIDlet is currently suspended or running in the background. When a MIDlet is suspended or runs in the background, all multimedia services will be blocked.

When in the Java Service Menu, the following apply when selected:

- Suspend – suspends the background MIDlet.
- Resume – brings the background MIDlet to the foreground and multimedia resources will be available for the MIDlet.
- End – destroys the background MIDlet.
- Run in background – lets the MIDlet continue to run in the background. Note: A Java icon will be displayed in the status bar.

7
Record Management System

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 16Kb will be used as the maximum RMS space for that MIDlet. No additional Motorola implementation clarifications are necessary.

Refer to the table below 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 fort 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 javax.microedition.rms package	Supported
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

8
JAD Attributes

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. The table below 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 500K.

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 suite	Yes	Yes
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.MIDletclass`.	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 J2ME 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 J2ME Configuration required, i.e CLDC 1.0	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 critical to the function of the MIDlet suite.	Yes	Yes
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

Background	MIDlets with this Motorola specific attribute will continue to run when not in focus.	Yes	Yes

9
LCDUI

LCDUI API

The following table 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 Item[286] objects
ItemStateListener	This interface is used by applications which need to receive events that indicate changes in the internal state of the interactive items within a Form[231] screen.

The following table 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 car be added to a Form[231].
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.

Refer to the following table 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 javax.microedition.lcdui package	Supported

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

10 Gaming API/Multiple Key Press

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.

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 Motorola C650 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 the table below 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 javax.microedition.lcdui.GameCanvas to the bottom position of the key	Supported
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	Transparency only

11
Vibe and Backlight API

Vibe and Backlight API

The Vibe and Backlight API allows J2ME applications access to vibrator, backlight, and keypad controls. This API gives a MIDlet the ability to turn these features on or off based on the applications needs. The MIDlet will use this API to enhance the activity being performed by the application.

Examples of this enhancement are the following:

1. When in a driving game application, the vibrator is turned on during a crash.
2. An alarm application would have access to turn the vibrator on and off.
3. A stock ticker application turns the backlight on and off when a specified stock hits a target price.

Native constraints are in place to protect the battery life of the product. These native constraints will be flexible to allow the operator to make the final call on implementation. For more information refer to the MIDP 2.0 specification.

The following are code samples to show implementation of the Vibe and Backlight API:

Sample of code for calling of 'vibrate(int)' method of Display class:

```
int duration = 3000;

returnValue = display.vibrate(duration);

if (returnValue != false) {
System.out.println("Invoke vibrate method with parameter = "
+ duration + ", method returns : " + returnValue);
}
else {
System.out.println("Failed: invoke vibrate(" + duration +
"), method returns false");
}
```

Sample of code for calling of 'flashBacklight(int)' method of Display class

```
int duration = 3000;

returnValue = display.flashBacklight(duration);
if (returnValue != false) {
System.out.println("Invoke flashBacklight method with
parameter = " + duration  + ", method returns : " +
returnValue);
}
else {
System.out.println("Failed: invoke flashBacklight(" +
duration + "), method returns false");
}
```

12
Fun Lights API

Fun Lights API

The Fun Lights API allows J2ME applications access to the Fun Lights feature on the Motorola C650. The Fun Lights feature gives a MIDlet the ability to turn the Fun Lights feature on or off based on the applications needs. The MIDlet would use this API to enhance the activity being performed by the application.

Examples of this enhancement would be the following:

- When in a driving game application, the Fun Lights turn on at varying light intensities and colors to warn of an impending crash.
- A karaoke or jukebox application turns the Fun Lights on and off to the beat of the music.
- An alarm clock or stopwatch application uses the Fun Lights in conjunction with a timer feature.

Fun Lights Regions

The Motorola C650 has areas that light up or change colors based on an assigned region number. By assigning priority ratings to the regions, all products have the ability to use Fun Lights. Querying the number of regions a handset has allows for Fun Light patterns to be mapped to a specific region. For example,

- A MIDlet will query the handset's native software to determine the number of light regions and from there, can map Fun Lights features to each of the regions.

The regions for the Motorola C650 are outlined below:

Region	Description
1	LCD Backlight

2	Keypad (LCD backlight will turn on by design)
4	Joystick LED

The Fun Lights features that J2ME MIDlets access will include individual LED control – color changes of LEDs (seven colors) and light intensity of LEDs (three levels). Motorola will create the color table for the 256 colors supported. If a color in a Fun Light feature is not supported by the handset, the closest color that matches the specified color shall be used.

Each of these LEDs can be set to on or off and due to human factor concerns the refresh rate will be less than 5 Hz.

The following are code samples to show implementation of the Fun Lights API:

Sample of operating regions one by one

```
/*
*  Obtain the array of regions
*/
Region regions = FunLight.getRegions();

/*
*  Set MAGENTA color for all regions
*/
for(int i = 0; i < regions.length; i++) {

    /*
    *  Obtain control of i's region
    */
    int oc_result = regions[i].getControl();

    /*
    *  Make sure control was obtained
    */
    if(oc_result != FunLight.SUCCESS) {
        /*
        *  Control was not obtained
        */
        if(oc_result == FunLight.QUEUED) {

            /*
            *  The request of obtaining region's control
was queued
            *  The control will be obtained when it
released by other application
            */
            ...
        }
        if(oc_result == FunLight.IGNORED) {

            /*
            *  The request of obtaining region's control
was ignored
```

```
                    *  It is related to 5Hz limit
                    */
                    ...
                }
            }

            /*
            *  Set MAGENTA color for i's region
            */
            int sc_result = regions[i].setColor(FunLight.MAGENTA);

            /*
            *  Make sure the color was set
            */
            if(sc_result != FunLight.SUCCESS) {
                /*
                *  The color was not set
                */
                if(sc_result == FunLight.QUEUED) {

                    /*
                    *  The request of setting region's color was
queued
                    *  (it means the MIDlet does not have control
of the region)
                    *  The color will be set when the control of
the regions will be obtained
                    */
                    ...
                }
                if(sc_result == FunLight.IGNORED) {

                    /*
                    *  The request of setting region's color was
ignored
                    *  It is connected with 5Hz limit

                    */
                    ...
                }
            }

            /*
            *  Obtain the color of the region
            */
            int color = regions[i].getColor();

            /*
            *  Check the color of the region
            */
            if(color != FunLight.MAGENTA) {

                /*
                *  Maybe it's a region which supports only white
color
                */
                if(color != FunLight.WHITE) {
```

```
            /*
            *  OK. We got a region which supports only
white color
            */
            ...
        } else {

            /*
            *  Error situation: the color differs from
MAGENTA and WHITE
            */
            ...
        }
    }

    /*
    *  Release the control of i's region
    */
    regions[i].releaseControl();

            }
```

Sample of operating regions all at once

```
/*
*  Obtain control of all regions
*/
int oc_result = FunLight.getControl();

/*
*  Make sure the control of all regions was obtained
*/
if(oc_result != FunLight.SUCCESS) {
    /*
    *  The control of at least one region was not obtained
    */
    if(oc_result == FunLight.QUEUED) {

        /*
        *  The request of obtaining region's control was
queued for at least one region
        */
        ...
    }
    if(oc_result == FunLight.IGNORED) {

        /*
        *  The request of obtaining region's control was
ignored for at least one region
        */
        ...
    }
}

/*
*  Set color for all regions
```

```
*/
int sc_result = FunLight.setColor(FunLight.MAGENTA);

/*
*  Make sure the color was set for all regions
*/
if(sc_result != FunLight.SUCCESS) {
    /*
    *  The color of at least one region was not set
    */
    if(sc_result == FunLight.QUEUED) {

        /*
        *  The request of setting region's color was queued
for at least one region
        */
        ...
    }
    if(sc_result == FunLight.IGNORED) {

        /*
        *  The request of setting region's color was
ignored for at least one region
        */
        ...
    }
}

/*
*  Release control of all regions
*/
FunLight.releaseControl();
```

13 Java.lang Implementation

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.

14
iTAP

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 J2ME 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 J2ME 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 J2ME 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 the table below 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 EMAILADDR, PASSWORD, or

URL

15
Network APIs

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
- Socket Support
- Datagram (UDP)

Refer to the table below 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	Supported
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

ServerSocketConnection 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
ServerSocketConnection interface in the javax.microedition.io.package	Supported
UDPDatagramConnection interface in the javax.microedition.io.package	Supported

The following is a code sample to show 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
                }
            }
```

User Permission

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

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 below.

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

HTTPS Connection

Motorola implementation supports a HTTPS connection on the Motorola C650 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://home.netscape.com/eng/ssl3/draft302.txt

The following is a code sample to show 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;
                }
```

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.

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.

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.

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.

The following is a code sample to show 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, TextField.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", ChoiceGroup.MULTIPLE,
      registeredConns, null);
                              if(registeredConnsCG.size() > 0)
      unregForm.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://:" +
regConnection.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.getStri
ng(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());
                  }
              }
        }
}
```

Delivery of a Push Message

A push message intended for a MIDlet on the Motorola C650 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 below.

- 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 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.

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.

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.

16
Interface CommConnection

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.

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>]`

Parameters

The first parameter will be a port identifier, which is a logical device name. These port identifiers are 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.

The table below 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.

BNF Format for Connector.open () string

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

<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="<baud_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"

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 implementing 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.

The following are code samples to 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;
```

```
        }
```

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.

Method Summary

The tables below 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

17
Platform Request API

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 the table below 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

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");
```

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.

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.

18
JSR 135 Mobile Media API

JSR 135 Mobile Media API

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

- Tone Sequence
- Sampled Audio
- MIDI

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

The following is a code sample to show 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();
```

ToneControl

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

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

The following is the available method for ToneControl:

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

VolumeControl

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

The following describes the different volume settings found within VolumeControl:

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

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

`-getLevel`: Get the current volume setting.

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

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

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

StopTimeControl

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

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

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

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

Manager Class

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

Audio Media

The following multimedia file formats are supported:

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

The following is a list of audio MIME types supported:

Category	Description	MIME Type
Audio	iMelody	`audio/imelody x-imelody imy x-imy`
	MIDI	`audio/midi x-midi mid x-mid sp-midi`
	WAV	`audio/wav x-wav`
	MP3	`audio/mp3 x-mp3 mpeg3 x-mpeg3 mpeg x-mpeg`
	AMR/MP4	`audio/amr x-amr mp4 x-mp4`

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

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

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

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

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

19
JSR 120 – Wireless Messaging API

Wireless Messaging API (WMA)

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

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

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

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

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

SMS Client Mode and Server Mode Connection

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

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

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

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

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

SMS Port Numbers

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

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

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

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

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

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

SMS Storing and Deleting Received Messages

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

SMS Message Types

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

SMS Message Structure

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

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

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

SMS Notification

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

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

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

The following 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

The following are code samples to show implementation of the JSR 120 Wireless Messaging API:

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

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

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

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

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

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

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

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

Creation of server connection:

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

Creation of client connection with port number:

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

Creation of client connection without port number:

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

Closing of connection:

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

Creation of SMS message:

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

Setting of payload text for text message:

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

Getting of payload text of received text message:

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

Getting of payload data of received binary message:

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

Setting of address with port number:

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

Setting of address without port number:

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

Sending of message:

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

Receiving of message:

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

Getting of address:

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

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

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

Getting of timestamp for the message:

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

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

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


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


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


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


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


byte[] newBin = createBinary(msgLength);
```

```
   binMsg.setPayloadData(newBin);

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

Setting of MessageListener and receiving of notifications about incoming messages:

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

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

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

public void run() {
      try {
      messageConnection.setMessageListener(listener);
      } catch (IOException e) {
         result = FAIL;
```

```
            System.out.println("FAILED: exception while setting
            listener: " + e.toString());
               }
            }
            }
```

20
Phonebook Access API

Phonebook Access API

Using the Phonebook Access API, an application will be able to locate and update contact information on the handset. This contact information includes phone numbers, email addresses, and any other directory information related to individuals, groups, or organizations. The database used to store phonebook information will be unique and integrated for native phonebook, SIM card, and other applications using Phonebook API.

The primary goal of the Phonebook Access API is to be simple and thin to fit in resource-limited devices like the Motorola C650 handset. This API will specify a base storage class for all types of contacts items presented in the vCard specification (RFC2426 –vCard MIME Directory Profile – vCard 3.0 Specification). In addition, schema strings used in querying and storing contact information are those specified in the RFC2426 specification.

The Phonebook Access API will perform the following functions:

- Support multiple phonebook categories
- Allow multiple phone numbers and email addresses for each contact
- Store new entries
- Retrieve entries
- Edit existing entries
- Delete entries
- Check memory status
- Order and sort contact parameters
- Support standard schema strings
- Support recent calls information

Phonebook Access API Permissions

Prior to a MIDlet accessing the Phonebook API for all Phonebook operations, the implementation will check the Phonebook permissions under the Java Settings Menu. The phonebook permissions menu gives the user the following options:

- Always ask the user for authorization on all Phonebook access requests
- Ask the user for authorization once per application (Default setting)
- Never ask the user for authorization

The following are code samples to show implementation of the Phonebook API:

Sample of code to create object of PhoneBookRecord class:

```
PhoneBookRecord phbkRecEmpty = new PhoneBookRecord();

String name = "Name";
String telNo = "9999999";
int type = PhoneBookRecord.MAIN;
int categoryId = PhoneBookRecord.CATEGORY_GENERAL;

PhoneBookRecord phbkRec = new PhoneBookRecord(name, telNo,
type, categoryId);
```

Sample of code for calling of 'add(int sortOrder)' method:

```
int index = phbkRec.add(PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'update(int index, int sortOrder)' method:

```
phbkRec.type = PhoneBookRecord.HOME;
int newIndex = phbkRec.update(index,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'delete(int index, int sortOrder)' method:

```
PhoneBookRecord.delete(index, PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'deleteAll()' method:

```
PhoneBookRecord.deleteAll();
```

Sample of code for calling of 'getRecord(int index, int sortOrder)' method:

```
phbkRec.getRecord(index, PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'findRecordByTelNo(String tel, int sortOrder)' method:

```
index = phbkRec.findRecordByTelNo(telNo,
```

```
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'findRecordByName(char firstChar, int sortOrder)' method:

```
index = PhoneBookRecord.findRecordByName('N',
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'findRecordByEmail(String email, int sortOrder)' method:

```
String email = "email@mail.com";
index = phbkRec.findRecordByEmail(email,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'getNumberRecords(int device)' method:

```
// get total number of records
int numberRecsInPhone =
PhoneBookRecord.getNumberRecords(PhoneBookRecord.PHONE_MEMOR
Y);
int numberRecsInSim =
PhoneBookRecord.getNumberRecords(PhoneBookRecord
.SIM_MEMORY);
int numberRecsAll =
PhoneBookRecord.getNumberRecords(PhoneBookRecord.ALL_MEMORY)
;
```

Sample of code for calling of 'getAvailableRecords(int device)' method:

```
// get number of available records
int numberRecsAvalPhone =
PhoneBookRecord.getAvailableRecords(PhoneBookRecord.PHONE_ME
MORY);
int numberRecsAvalSim =

PhoneBookRecord.getAvailableRecords(PhoneBookRecord.SIM_MEMO
RY);
int numberRecsAvalAll =
PhoneBookRecord.getAvailableRecords(PhoneBookRecord.ALL_MEMO
RY);
```

Sample of code for calling of 'getUsedRecords(int device, int sortOrder)' method:

```
// get number of used records
int numberRecsUsedPhone =
PhoneBookRecord.getUsedRecords(PhoneBookRecord.PHONE MEMORY,
PhoneBookRecord.SORT_BY_NAME);
int numberRecsUsedSim =
PhoneBookRecord.getUsedRecords(PhoneBookRecord.SIM_MEMORY,
PhoneBookRecord.SORT_BY_NAME);
int numberRecsUsedAll =
PhoneBookRecord.getUsedRecords(PhoneBookRecord.ALL_MEMORY,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'getNumberRecordsByName(String name)' method:

```
int num = PhoneBookRecord.getNumberRecordsByName(name);
```

Sample of code for calling of 'getMaxNameLength(int device)' method:

```
int maxNameLengthPhone =
PhoneBookRecord.getMaxNameLength(PhoneBookRecord.PHONE_MEMOR
Y);
int maxNameLengthSim =
PhoneBookRecord.getMaxNameLength(PhoneBookRecord.SIM_MEMORY)
;
int maxNameLengthAll =
PhoneBookRecord.getMaxNameLength(PhoneBookRecord.ALL_MEMORY)
;
```

Sample of code for calling of 'getMaxTelNoLength (int device)' method:

```
int maxTelNoLengthPhone =
PhoneBookRecord.getMaxTelNoLength(PhoneBookRecord.PHONE_MEMO
RY);
int maxTelNoLengthSim =
PhoneBookRecord.getMaxTelNoLength(PhoneBookRecord.SIM_MEMORY
);
int maxTelNoLengthAll =
PhoneBookRecord.getMaxTelNoLength(PhoneBookRecord.ALL_MEMORY
);
```

Sample of code for calling of 'getMaxEmailLength ()' method:

```
int maxEmailLength =
PhoneBookRecord.getMaxEmailLength();
```

Sample of code for calling of 'getIndexBySpeedNo(int speedNo, int sortOrder)' method:

```
int speedNo = 1;
index = PhoneBookRecord.getIndexBySpeedNo(speedNo,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'getNewSpeedNo(int num, int device)' method:

```
int speedNo = 1;
int speedNo_phone =
PhoneBookRecord.getNewSpeedNo(speedNo,
PhoneBookRecord.PHONE_MEMORY);
int speedNo_sim =
PhoneBookRecord.getNewSpeedNo(speedNo,
 PhoneBookRecord.PHONE_MEMORY);
int speedNo_all =
PhoneBookRecord.getNewSpeedNo(speedNo,
PhoneBookRecord.PHONE_MEMORY);
```

Sample of code for calling of 'getDeviceType(int speedNo)' method:

```
int speedNo = 1;
int type = PhoneBookRecord.getDeviceType(speedNo);
```

Sample of code for calling of 'setPrimary(int index, int sortOrder)' method:

```
int index = 1;
PhoneBookRecord.setPrimary(index,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'resetPrimary(int index, int sortOrder)' method:

```
int index = 1;
PhoneBookRecord.resetPrimary(index,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'isPrimary(int speedNo)' method:

```
int speedNo = 1;
boolean res = PhoneBookRecord.isPrimary(speedNo);
```

Sample of code for calling of 'fromVFormat(InputStream in, int device)' method:

```
buffer = new String("BEGIN:VCARD\r\nN:;" + new String(name)
+
"\r\nTEL;TYPE=WORK:1\r\nEND:VCARD\r\n");
int num =
PhoneBookRecord.fromVFormat((InputStream)(new
ByteArrayInputStream(buffer.getBytes())),
PhoneBookRecord.PHONE_MEMORY);
```

```
Sample of code for calling of 'toVFormat(OutputStream out,
int index, int outFormat, int sortOrder)' method:
```

```
int index = 1;
ByteArrayOutputStream outStream = new
ByteArrayOutputStream();
PhoneBookRecord.toVFormat(outStream, index,
PhoneBookRecord.VCARD_3_0,
PhoneBookRecord.SORT_BY_NAME);


System.out.println("***** Contents of the output stream:
*****");
System.out.print(new String(outStream.toByteArray()));
```

Sample of code for calling of 'createMailingList(int[] members, int sortOrder)' method:

```
PhoneBookRecord mailingList = new PhoneBookRecord();
int mlSpeedNumbers[] = new int[2];
mlSpeedNumbers[0] = 1;
mlSpeedNumbers[1] = 2;
```

```
mailingList.name = "MList";
mailingList.type = PhoneBookRecord.MAILING_LIST;
mailingList.speedNo =
 PhoneBookRecord.getNewSpeedNo(1,
PhoneBookRecord.PHONE_MEMORY);

index = mailingList.createMailingList(mlSpeedNumbers,
PhoneBookRecord.SORT_BY_NAME);
```

Sample of code for calling of 'addMailingListMember(int mlSpeedNo, int mbSpeedNo)' method:

```
int mlspeedNo = 3, mbspeedNo = 4;
PhoneBookRecord.addMailingListMember(mlspeedNo, mbspeedNo);
```

Sample of code for calling of 'deleteMailingListMember(int mlSpeedNo, int mbSpeedNo)' method:

```
int mlspeedNo = 3, mbspeedNo = 4;
PhoneBookRecord.deleteMailingListMember(mlspeedNo,
mbspeedNo);
```

Sample of code for calling of 'getMailingListMembers(int speedNo)' method:

```
int mlspeedNo = 3;
int[] returnArray =
PhoneBookRecord.getMailingListMembers(mlspeedNo);
```

Sample of code for calling of 'isMailingListMember(int mlSpeedNo, int mbSpeedNo)' method:

```
boolean returnValue = false;
int mlspeedNo = 3, mbspeedNo = 4;
returnValue = PhoneBookRecord.isMailingListMember(mlspeedNo,
mbspeedNo);
```

Sample of code for calling of 'getNumberMailingListMembers(int speedNo)' method:

```
int numberMembers, mlspeedNo = 3;
numberMembers =
PhoneBookRecord.getNumberMailingListMembers(mlspeedNo);
```

Sample of code for calling of 'addCategory(String name)' method:

```
String categoryName = "CatName";
int categoryId = PhoneBookRecord.addCategory(categoryName);
```

Sample of code for calling of 'deleteCategory(int categoryId)' method:

```
PhoneBookRecord.deleteCategory(categoryId);
```

Sample of code for calling of 'getCategoryName(int categoryId)' method:

```
String categoryName =
PhoneBookRecord.getCategoryName(categoryId);
```

Sample of code for calling of 'getCategoryMembers(int categoryId)' method:

```
int SpeedNumbersArray[] = null;
SpeedNumbersArray =
PhoneBookRecord.getCategoryMembers(categoryId);
```

Sample of code for calling of 'getNumberCategoryMembers (int categoryId)' method:

```
int numberMembers =
PhoneBookRecord.getNumberCategoryMembers(categoryId);
```

Sample of code for calling of 'getNumberCategories()' method:

```
int numberCategories =
PhoneBookRecord.getNumberCategories();
```

Sample of code for calling of 'getCategoryIdByIndex(int index)' method:

```
int index = 1;
int categoryId =
PhoneBookRecord.getCategoryIdByIndex(index);
```

Sample of code for calling of 'getMaxCategoryNameLength()' method:

```
int maxCategoryNameLength =
PhoneBookRecord.getMaxCategoryNameLength();
```

Sample of code for calling of 'getCurrentCategoryView()' method:

```
int categoryView = PhoneBookRecord.getCurrentCategoryView();
```

Sample of code for calling of 'setCategoryView()' method:

```
int oldCategoryView =
PhoneBookRecord.setCategoryView(categoryId);
```

Sample of code to create object of RecentCallDialed class:

```
String name = "Name";
String telNo = "9999999";
int type = RecentCallRecord.VOICE;
int attribute = RecentCallRecord.CALL_CONNECTED;
long time = 10000;
int duration = 3000;
boolean show_id = true;

RecentCallDialed dialedRecentCall = new
RecentCallDialed(name, telNo, type, attribute, time,
duration, show_id);
```

Sample of code for calling of 'add()' method:

```
String name = "Name";
String telNo = "9999999";
int type = RecentCallRecord.VOICE;
int attribute = RecentCallRecord.CALL_CONNECTED;
long time = 10000;
int duration = 3000;
boolean show_id = true;

RecentCallDialed dialedRecord = new RecentCallDialed(name,
telNo, type, attribute, time, duration, show_id);
dialedRecord.add();
```

Sample of code for calling of 'delete(int index)' method:

```
int index = 1;
RecentCallDialed.delete(1);
```

Sample of code for calling of 'deleteAll()' method:

```
RecentCallDialed.deleteAll();
```

Sample of code for calling of 'getRecord(int index)' method:

```
int index = 1;
dialedRecord.getRecord(1);
```

Sample of code for calling of 'getUsedRecords()' method:

```
int usedRecs = RecentCallDialed.getUsedRecords();
```

Sample of code for calling of 'getNumberRecords()' method:

```
int numberRecs = RecentCallDialed.getNumberRecords();
```

Sample of code for calling of 'getMaxNameLength()' method:

```
int maxNameLength = RecentCallDialed.getMaxNameLength();
```

Sample of code for calling of 'getMaxTelNoLength()' method:

```
int maxTelNoLength = RecentCallDialed.getMaxTelNoLength();
```

Sample of code to create object of RecentCallReceived class:

```
String name = "Name";
String telNo = "9999999";
int type = RecentCallRecord.VOICE;
int attribute = RecentCallRecord.CALL_CONNECTED;
long time = 10000;
int duration = 3000;
int cli_type  = RecentCallReceived.CALLER_ID_NAME;
```

```
RecentCallReceived receivedRecentCall = new
RecentCallReceived (name, telNo, type, attribute, time,
duration, cli_type);
```

Sample of code for calling of 'add()' method:

```
String name = "Name";
String telNo = "9999999";
int type = RecentCallRecord.VOICE;
int attribute = RecentCallRecord.CALL_CONNECTED;
long time = 10000;
int duration = 3000;
int cli_type  = RecentCallReceived.CALLER_ID_NAME;

RecentCallReceived receivedRecord = new
RecentCallReceived(name, telNo, type, attribute, time,
duration, show_id);
receivedRecord.add();
```

Sample of code for calling of 'delete(int index)' method:

```
int index = 1;
RecentCallReceived.delete(1);
```

Sample of code for calling of 'deleteAll()' method:

```
RecentCallReceived.deleteAll();
```

Sample of code for calling of 'getRecord(int index)' method:

```
int index = 1;
receivedRecord.getRecord(1);
```

Sample of code for calling of 'getUsedRecords()' method:

```
int usedRecs = RecentCallReceived.getUsedRecords();
```

Sample of code for calling of 'getNumberRecords()' method:

```
int numberRecs = RecentCallReceived.getNumberRecords();
```

Sample of code for calling of 'getMaxNameLength()' method:

```
int maxNameLength = RecentCallReceived.getMaxNameLength();
```

Sample of code for calling of 'getMaxTelNoLength()' method:

```
int maxTelNoLength = RecentCallReceived.getMaxTelNoLength();
```

21
File System Access API

The File System Access API for the Motorola C650 handset is intended to allow J2ME applications access to files that are stored on the handsets. Internal file systems in RAM or ROM can be accessed via the File System Access API. Please note that the maximum file name length supported by Motorola is 36 characters.

File Connection Methods

The File System Access API will provide file connections provided there is underlying hardware and OS support on the handset. If file connections are not supported, attempts to open a file connection through `Connector.open()` will throw `javax.micdoedition.io.ConnectionNotFoundException.`

Establishing a Connection

To establish a connection, the format of the `Connector.open()` input string used to access a File Connection will follow the format described in the file URL format as part of IETF RFCs 1738 and 2386. These IETF RFCs dictate that a file URL will take the following form:

- `File//<host>/<path>`

In the format shown above, the following definitions will be used:

- <host> - fully qualified domain name of the system on which the `<path>` is accessible
- <path> - a hierarchical directory path of the form `<root>/<directory>/<directory>/…/<name>/`

For successful implementation, `<host>` will be the empty string. This is read as the machine from which the URL is being interpreted. This type of file connection only supports file access to files contained on a device or one of its attached memory cards. This type of file connections does not support access to files on a remote file system. Attempts to open a remote file system connection with `Connector.open()` will throw a `javax.microedition.io.IOException`.

The file connection defines the first directory as the root, which corresponds to a logical mount point for a particular storage unit or memory. The root of a file system supporting only one card reader will return the contents of the card directly. If more than one card reader is supported, or the device supports a file system in RAM, the root strings are device specific. Since roots are part of the path string as a directory, they have a trailing "/". The valid `<root>` values for a device can be retrieved by querying open a root connection to the root URL of the device (`file:///`) and getting the results of the list() method. A single connection object can only reference a single file or directory. A connection object cannot be changed dynamically to refer to a different file or directory than originally connected to. A completely separate connection will be established through the `Connector.open()` method to reference a different file or directory.

Connection Behavior

The File System Access API is different from other Generic Connection Framework APIs because a connection object can successfully return from the `Connector.open()` method without referencing an existing file or directory. This method is used to allow the creation of new files and directories on a file system. Always check for the file's or directory's existence after a connection is established to determine if the file or directory actually exists.

Similarly, files or directories can be deleted using the `delete()` method. The connection should immediately be closed after a deletion to prevent exceptions from accessing a connection to a non-existent file or directory.

Security

To prevent unauthorized manipulation of data, access to file connections is restricted. The security model applied to the file connection is defined by the implementing profile. This security model will be applied on the invocation of the `Connector.open()` method with a valid file connection string. Should the application not be granted access to the file system 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.

File access through the File System Access API is restricted in order to protect the user's files and data from malicious and unintentional access. A file connection will not be able to access RMS databases, files that are private to another application, system configuration files, and device and OS specific files.

Please note that only signed applications residing in manufacturer and operator domains will have access to the File System.

The following are code samples to show implementation of the File System API:

Sample for FileConnection Interface

```
import javax.microedition.io.*;
import com.motorola.io.*;

/*
*  Get a root of file system
*/
java.lang.String root = FileSystemRegistry.listRoots()[0];

/*
* Create a FileConnection object pointing to root directory
*/
 FileConnection fc = null;
try {
       fc = (FileConnection)
Connector.open("file://"+root);
} catch (IOException ioe) {
            …
}

/*
* Get available, used and total size of file system
*/

long availableSize = fc.availablesize();

long usedSize = fc.usedSize();

long totalSize = fc.totalSize();


/*
* Determine the size in bytes that is used by a directory
*/

try {
      long rootSize = fc.directorySize();
} catch (IOException ioe) {
            …
}

/*
* Get list of files in root
*/

java.lang.String list[] = fc.list();

/*
* Check if first item in the list is directory
*/

try {
       fc = (FileConnection)
Connector.open("file://"+list[0]);
} catch (IOException ioe) {
            …
```

```
}

if (fc.isDirectory()) {
      // first item is directory
}
else {
      //first item is file
}

/*
* Create a FileConnection object pointing to file
*/
 FileConnection fc = null;
try {
       fc = (FileConnection)
Connector.open("file://"+root+"testfile.txt");
} catch (IOException ioe) {
            …
}

/*
* If file does not exists, create it
*/

if (! fc.exists() ) {
      try {

             fc.create();
      } catch (SecurityException se) {
            ...
      }
}

/*
*  Open stream for writing data
*/

DataOutputStream dataOutputStream;
try {
      dataOutputStream = fc.openDataOutputStream();
      //write data
      dataOutputStream.close();
} catch (IOException ioe) {
            …
}

/*
*  Open stream for reading data
*/

DataInputStream dataInputStream;
try {
      dataInputStream = fc.openDataInputStream();
      // read data
      dataInputStream.close();
} catch (IOException ioe) {
            …
```

```
}

/*
* Get file size
*/

try {
      long fileSize = fc.fileSize();
} catch (IOException ioe) {
            …
}

/*
* Set file's attributes
*/

try {
      boolean isSetToWrite = fc.setWriteable(true);
} catch (IOException ioe) {
            …
}

try {
      boolean isSetToRead = fc.setReadable(true);
} catch (IOException ioe) {
            …
}

try {
      boolean isSetHidden =fc.setHidden(true);
} catch (IOException ioe) {
            …
}

/*
* Get file's attributes
*/

try {
      boolean isWriteable = fc.canWrite();
} catch (IOException ioe) {
            …
}
try {
      boolean isReadable =fc.canRead();
} catch (IOException ioe) {
            …
}
try {
      boolean isHidden =fc.isHidden();
} catch (IOException ioe) {
            …
}

/*
* Rename file
*/
```

```
boolean isRenamed =
fc.rename("file://"+root+"newfile.txt");

/*
* Get path and URL of file
*/

java.lang.String filePath = fc.getPath();
java.lang.String fileURL = fc.getURL();

/*
* Get time of last modification
*/

long lastModifiedTime = fc.lastModified();

/*
* Delete file

*/

boolean isDeleted = fc.delete();

/*
* Close File Connection
*/

try {
      fc.close();
} catch (IOException ioe) {
            …
}
```

FileSystemRegistry Class

```
/*
* Get list of roots
*/

java.lang.String listOfRoot =
FileSystemRegistry.listRoots();

/*
*  To add/remove file system listener it is necessary to
create an instance of FileSystemListener interface:
*/

Listener newListener = new Listener();

/*
* add listener
*/
FileSystemRegistry.addFileSystemListener(listener);

/*
```

```
* remove listener
*/

FileSystemRegistry.removeFileSystemListener(listener);

/* Implementation of FileSystemListener interface
*
* Method rootAdded(FileSystemEvent e) is invoked when a
root is added to the filesystem
* Method rootRemoved(FileSystemEvent e) is invoked when a
root is removed from the filesystem
* To get information about just added/removed roots, the
following methods of FileSystemEvent should be used:
* - to get event ID:    e.getID();
* - to get root name:   e.getRootName();
*/
class Listener implements FileSystemListener {

      Listener() {
      }
      public void rootAdded(FileSystemEvent e) {
            //implementation of the method
      }
      public void rootRemoved(FileSystemEvent e) {
            //implementation of the method
      }

   }
```

22
MIDP 2.0 Security Model

The following sections describe the MIDP 2.0 Default Security Model for the Motorola C650 handset. The chapter discusses the following topics:

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

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

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

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

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

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

Untrusted MIDlet Suites

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

The following are conditions of untrusted MIDlet suites:

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

Untrusted Domain

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

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

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

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

Trusted MIDlet Suites

Trusted MIDlet suites are MIDlet suites in which the integrity of the JAR file can be authenticated and trusted by the device, and bound to a protection domain. The Motorola C650 will use x.509PKI for signing and verifying trusted MIDlet suites.

Security for trusted MIDlet suites will utilize protection domains. Protection domains define permissions that will be granted to the MIDlet suite in that particular domain. A MIDlet suite will belong to one protection domain and its defined permissible actions. For implementation on the Motorola C650, the following protection domains should exist:

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

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

Permission Types concerning the Handset

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

- “Allowed” (Full Access) permissions are any permissions that explicitly allow access to a given protected API or function from a protected domain. Allowed permissions will not require any user interaction.
- “User” permissions are any permissions that require a prompt to be given to the user and explicit user confirmation in order to allow the MIDlet suite access to the protected API or function.

User Permission Interaction Mode

User permission for the Motorola C650 handset is designed to allow the user the ability to either deny or grant access to the protected API or function using the following interaction modes (bolded term(s) is prompt displayed to the user):

- blanket – grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is uninstalled or the permission is changed by the user. (Never Ask)
- session – grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is terminated. This mode will prompt the user on or before the final invocation of the protected API or function. (Ask Once Per App)
- oneshot – will prompt the user each time the protected API or function is requested by the MIDlet suite. (Always Ask)
- No – will not allow the MIDlet suite access to the requested API or function that is protected. (No Access)

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

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

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

Implementation based on Recommended Security Policy

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

Trusted 3rd Party Domain

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

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

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

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

The following table 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

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

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

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

Security Policy for Protection Domains

The following table lists the security policy by function groups for each domain. Under each domain are the settings allowed for that function within the given domain, while the bolded setting is the default setting. The Function Group is what will be displayed to the

user when access is requested and when modifying the permissions in the menu. The default setting is the setting that is effective at the time the MIDlet suite is first invoked and remains in effect until the user changes it.

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

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

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

MIDP 2.0 Specific Functions		
Permission	Protocol	Function Group
javax.microedition.io.Connector.http	http	Data Network
javax.microedition.io.Connector.https	https	Data Network

javax.microedition.io.Connector.datagram	datagram	Data Network
javax.microedition.io.Connector.datagramreceiver	datagram server (w/o host)	Data Network
javax.microedition.io.Connector.socket	socket	Data Network
javax.microedition.io.Connector.serversocket	server socket (w/ o host)	Data Network
javax.microedition.io.Connector.ssl	ssl	Data Network
javax.microedition.io.Connector.comm	comm	Connectivity Options
javax.microedition.io.PushRegistry	All	App Auto-Start
Phonebook API		
com.motorola.phonebook.readaccess	PhoneBookRecord.findRecordByName() PhoneBookRecord.findRecordByTelNo() PhoneBookRecord.findRecordByEmail() PhoneBookRecord.getNumberRecordsByName() PhoneBookRecord.getRecord() PhoneBookRecord.toVFormat() PhoneBookRecord.getCategoryName() PhoneBookRecord.getMailingListMembers() RecentCallDialed.getRecord() RecentCallReceived.getRecord()	User Data Read Capability
com.motorola.phonebook.writeaccess	PhoneBookRecord.add() PhoneBookRecord.update() PhoneBookRecord.delete() PhoneBookRecord.deleteAll() PhoneBookRecord.setPrimary() PhoneBookRecord.resetPrimary() PhoneBookRecord.fromVFormat() PhoneBookRecord.addCategory() PhoneBookRecord.deleteCategory() PhoneBookRecord.setCategoryView() PhoneBookRecord.createMailingList()	User Data Write Capability

	PhoneBookRecord.addMailingListMember() PhoneBookRecord.deleteMailingListMember() RecentCallDialed.add() RecentCallDialed.delete() RecentCallDialed.deleteAll()	
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

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

Displaying of Permissions to the User

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

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

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

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

Trusted MIDlet Suites Using x.509 PKI

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

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

Signing a MIDlet Suite

The default security model involves the MIDlet suite, the signer, and public key certificates. A set of root certificates are used to verify certificates generated by the signer. Specially designed certificates for code signing can be obtained from the manufacturer, operator, or certificate authority. Only root certificates stored on the handset will be supported by the Motorola C650 handset.

Signer of MIDlet Suites

The signer of a MIDlet suite can be the developer or an outside party that is responsible for distributing, supporting, or the billing of the MIDlet suite. The signer will have a public key infrastructure and the certificate will be validated to one of the protection domain root certificates on the handset. The public key is used to verify the signature of JAR on the MIDlet suite, while the public key is provided as a x.509 certificate included in the application descriptor (JAD).

MIDlet Attributes Used in Signing MIDlet Suites

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

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

requested in the HAD are different than those requested in the manifest, the installation 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.

Creating the Signing Certificate

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

Inserting Certificates into JAD

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

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

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

Note the following:

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

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

Creating the RSA SHA-1 signature of the JAR

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

It will be noted that the signer of the MIDlet suite is responsible to its protection domain root certificate owner for protecting the domain's APIs and protected functions; therefore,

the signer will check the MIDlet suite before signing it. Protection domain root certificate owners can delegate signing MIDlet suites to a third party and in some instances, the author of the MIDlet.

Authenticating a MIDlet Suite

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

Verifying the Signer Certificate

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

1. Get the certification path for the signer certificate from the JAD attributes MIDlet-Certificate-1<m>, where <m> starts a 1 and is incremented by 1 until there is no attribute with this name. The value of each attribute is abase64 encoded certificate that will need to be decoded and parsed.
2. Validate the certification path using the basic validation process as described in RFC2459 using the protection domains as the source of the protection domain root certificates.
3. Bind the MIDlet suite to the corresponding protection domain that contains the protection domain root certificate that validated the first chain from signer to root.
4. Begin installation of MIDlet suite.
5. If attribute MIDlet-Certificate-<n>-<m> with <n> being greater than 1 are present and full certification path could not be established after verifying MIDlet-Certificate-<1>-<m> certificates, then repeat step 1 through 3 for the value <n> greater by 1 than the previous value.

The following table 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.

Verifying the MIDlet Suite JAR

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

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

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

Initial State	Verification Result
JAD not present, JAR downloaded	Authentication can not be performed, will install JAR. MIDlet suite is treated as untrusted. The following error prompt will be shown, "Application installed, but may have limited functionality."
JAD present but is JAR is unsigned	Authentication can not be performed, will install JAR. MIDlet suite is treated as untrusted. The following error prompt will be shown, "Application installed, but may have limited functionality."
JAR signed but no root certificate present in the keystore to validate the certificate chain	Authentication can not be performed. JAR installation will not be allowed. The following error prompt will be shown, "Root certificate missing. Application not installed."
JAR signed, a certificate on the path is expired	Authentication can not be completed. JAR installation will not be allowed. The following error prompt will be shown, "Expired Certificate. Application not installed."
JAR signed, a certificate rejected for reasons other than expiration	JAD rejected, JAR installation will not be allowed. The following error prompt will be shown, "Authentication Error. Application not installed."

JAR signed, certificate path validated but signature verification fails	JAD rejected, JAR installation will not be allowed. The following error prompt will be shown, “Authentication Error. Application not installed.”
Parsing of security attributes in JAD fails	JAD rejected, JAR installation will not be allowed. The following error prompt will be shown, “Failed Invalid File.”
JAR signed, certificate path validated, signature verified	JAR will be installed. The following prompt will be shown, “Installed.”

Carrier Specific Security Model

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

Appendix A: Key Mapping

Key Mapping for the C650

The table below 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.

The following table identifies keys that will be assigned to game actions defined in GameCanvas class of MIDP 2.0.

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

Appendix B: Memory Management Calculation

Available Memory

The available memory on the Motorola C650 handset is the following:

- 1.8MB shared memory for MIDlet storage
- 800 Kb Heap size
- Recommended maximum MIDlet size is 100K

Memory Calculation for MIDlets

The calculation for determining the amount of memory needed to run a MIDlet is computed by a formula. The details menu for the application show the Kilobytes required by the application as computed by the formula below:

(size of the JAD file)

+ (size of the JAR file)

+ (size of the data space used by the MIDlet)

==========

(memory required to run the MIDlet)

Please note that the same memory calculation is applied while performing the memory check during the download of an application.

Appendix C: FAQ

Online FAQ

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

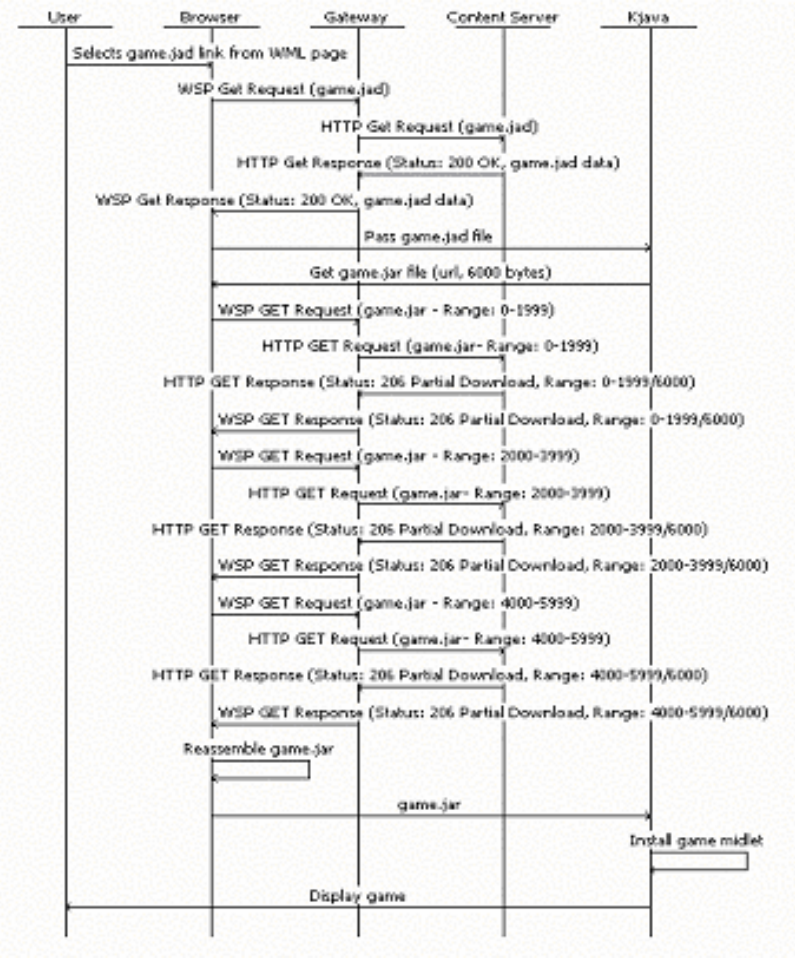
Access to dynamic content based on questions from the Motorola J2ME developer community is available at the URL listed below.

http://www.motocoder.com

Appendix D: HTTP Range

Graphic Description

The following is a graphic description of HTTP Range:



Appendix E: Spec Sheet

C650 Spec Sheet

Listed below are the spec sheets for the Motorola C650 handset. The spec sheet contains information regarding the following areas:

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

MOTOROLA
intelligence everywhere™

Motorola C650

Developer Reference Sheet

Technical Specifications

Band/Frequency	GSM 900/1800/1900 GPRS GSM 850/ 900/1900 GPRS
Region	Global
Technology	WAP 2.0, J2ME, SMS, EMS, MMS, AOL/OICQ IM
Connectivity	Mini-USB
Dimensions	83.5 x 44 x 22.2
Weight	95 g
Display	Internal: 128 x 128
Operating System	Motorola
Chipset	i250S1

Key Features

- Tri band
- Integrated digital camera (VGA quality)
- Stylish design with soft touch paint
- 1.8 MB of user memory
- Large, active color display (128 x 128)
- Games (embedded and downloadable)
- PIM functionality with Picture Caller ID
- Downloadable themes (ringers, images, sounds)
- High quality ring tones, MP3, and MIDI supported
- Voice memo
- 22 KHz polyphonic speaker with 24 chord support
- WAP 2.0

J2ME™ Information

CLDC v1.0 and MIDP v2.0 compliant	
Maximum MIDlet suite size	100Kb
Heap size	800 Kb
Maximum record store size	64K
MIDlet storage available	1.8 MB
Interface connections	HTTP, Socket, UDP, Serial port
Maximum number of sockets	4
Supported image formats	.PNG, .JPEG
Double buffering	Supported
Encoding schemes	ISO8859_1, ISO10646
Input methods	Multitap, iTAP
Additional API's	JSR 120, JSR 135, Phonebook
Audio	MIDI, WAV, AMR

Related Information

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

Tools:
CodeWarrior® Wireless Studio v7.0
J2ME™ SDK version v4.0
Motorola Messaging Suite v1.1

Documentation:
Creating Media for the Motorola C650

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

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