MOTODEV

The Motorola developer network

Java ME Developer Guide for Motorola OS

1.2

Developer Guide

Java ME Developer Guide for Motorola OS

Version 1.2

March 2008

For the latest version of this document, visit http://developer.motorola.com.

Motorola, Inc.

http://www.motorola.com

Contents

Chapter 1: Overview

Purpose and audience

This guide provides useful information for developers who want to develop Java™ ME (Micro Edition) applications—known as MIDlets—for Motorola handsets running Motorola OS. It includes information on support APIs, details on developing and packaging applications for installation, as well as a step-by-step procedure for setting up a debug environment. It does not teach you about Java ME or provide basics on developing Java ME applications; we assume you already know how to do that.

This document is intended for application developers who are already familiar with Java ME development and want to know how to develop MIDlets for Motorola handsets.

Developer tools

There are two tools available for developers, MOTODEV Studio for Java ME and MOTODEV SDK for Java ME.

MOTODEV Studio for Java ME

MOTODEV Studio for Java ME is a Java ME development environment for Motorola mobile devices. It is an extension of the Eclipse platform integrated with EclipseME, Motorola Java ME SDK, and the Motorola Update Manager. MOTODEV Studio is a robust, integrated development environment that gives you a fast and easy way to create applications that take advantage of the latest functionality in a wide array of Motorola products. Motorola's Java Emulator tool enables third-party developers to create Java applications for mobile devices. MOTODEV Studio for Java ME features easy "all in one place" access to Java ME libraries, sample MIDlets and tutorials, and integrated documentation.

Download MOTODEV Studio for Java ME at: https://developer.motorola.com/docstools/motodevstudio/.

MOTODEV SDK for Java ME

MOTODEV SDK for Java ME contains tools for developing and testing applications written in the Java programming language for Motorola handsets. This SDK supports handsets running either the Linux OS or the Motorola OS through the Motorola Java ME device emulator. The SDK is designed for use with UEI-compliant IDEs such as NetBeans and JBuilder, and has a built-in update manager. If you wish to use the Eclipse IDE, please use MOTODEV Studio for Java ME. For optional audio and other specialized requirements as well as a full list of features, known issues, and related information, see the Release Notes. Check the developer web site for the latest version of the SDK: https://developer.motorola.com/docstools/sdks/.

Additional resources

Many documentation resources are available to Motorola developers, including those that follow.

Technical articles

Technical articles are created regularly on a wide variety of topics related to Java ME development on Motorola OS handsets. All technical articles are posted on the Motorola developer web site available online at http://developer.motorola.com/docstools/technicalarticles/

The following table groups the technical articles into general topic categories, not necessarily applicable to all operating systems.

Table 1: MOTODEV Technical Articles

General Category	Technical Articles
Browsing	• Browser and Over-the Air Provisioning • User-Agent Profiles and User-Agent Strings • Basic Over-the-Air Server Configuration • Motorola Generic WAP Developer Style Guide
Connectivity	• Using HTTP and HTTPS on Motorola MIDP 2.0 Handsets • Using Serial Connections on Motorola Java ME Handsets
Developer Tools	• An Introduction to MOTODEV Studio • Using the MOTODEV SIMConfig Tool • NetBeans IDE and the Motorola Java ME SDK • Using the Motorola MIDway Tool • Installing MIDlets Using MIDway • Integrating Motorola's Lightweight Windowing Toolkit with Java ME Developer Tools
Device Management	• Provisioning with the Open Mobile Alliance Device Management Platform • Creating Device Configurations
DRM	Introduction of Basic Concepts in OMA DRM
Games	• A Simple Demo of Mobile Game Programming on the A1200 Handset • Performance Improvement Tips in M3G Games • 2D Game Programming for the Motorola V30, V400 and V500 Handsets
Image API	The Motorola Scalable JPEG Image API (deprecated, replaced by Motorola Scalable Image APIs)

Table 1: MOTODEV Technical Articles (Continued)

General Category	Technical Articles
Java ME	• Using JAD Attributes • Using hideNotify and showNotify on Motorola OS Handsets • Building J2ME Web Services Applications with the MOTOMING A1200 • Using Bluetooth on Motorola Handsets • Password Based Encryption in Java ME • Sharing Record Stores in MIDlet Suites • XML in Java ME • Introduction of MVC Structure in Java ME Clients • Using the Push Registry in MIDP 2.0 • Telephony • Threading in Java ME (MIDP 2.0)
Java ME	• Using Push Registry on Motorola Handsets • Using RMS on Motorola Java-Enabled Handsets • Motorola Custom Attributes in JAD Files • Implementing Key and Pointer Events via Java ME MIDlets on Motorola Symbian Handsets • Porting Java ME Applications from the T720i to the V300, V400, V500 and V600 • Porting a MIDlet from the i95cl to the T720
Java ME: Language Topics	• Chinese Character Encoding/Decoding in Java ME • Language Translation in Java ME Applications (MIDlets) • Handling of Right-To-Left Languages in Motorola's MIDP 2.0 J2ME Implementation • Motorola Language API for Java Applications
Java ME: Motorola-specific APIs	• Morphing Support • MIDlet Lifecycle on Motorola Linux OS Devices • Interaction of the MIDlet Life Cycle and Hot Execution Environment • Secondary Display API • Vibrate, Backlight and Fun Light APIs on Linux OS Motorola Handsets • Using Fun Lights • Using Backlight
Messaging	• Using the WMA Test Server for MMS Messaging • Introduction of MMS in Java ME • The Wireless Messaging API • Creating a WAP Email Client Using Perl

Table 1: MOTODEV Technical Articles (Continued)

General Category	Technical Articles
Multimedia	• Capturing Images and Video • 3D Programming - Loading M3G Files and Playing Animations • Transparent Images in MIDP 2.0 • The Java ME Mobile Media API (JSR-135) • Mobile 3D Graphics Programming • Troubleshooting Sound Player Issues on the E680/A780 • Image Capture for the V980 and E1000 Handsets • Sound Implementation on the V300, V500 and V600 • Using Sound on the Motorola V300, V500 and V600 Handsets • Graphics Programming on the Motorola V300, V500 and V600
Optimizing	• Optimizing a Java ME Application Part 3: Canvas Performance Improvement • Optimizing a Java ME Application Part 2: RMS Sorting • Optimizing a Java ME Application Part 1: Speed
Personal Identification	• JSR 75: Personal Information Management Redesign and Enhancement • The FileConnection API • Using PIM API to Import/Export vCards • Using JSR 75 (Personal Information Management)
Security	• Proper Speed and Heading Calculation Using Location Services • How to Set Up Your SSL Connection in Linux Devices
Testing and Debugging	• Debugging MIDlets on the MOTOSLVR L7 • Using KDWP to Debug MIDlets Running on Motorola Handsets
Windows Mobile	Programming the Motorola Q Windows Mobile Smartphone

Developer knowledge base

Developer Technical Support (DTS) has an extensive Frequently Asked Questions (FAQ) developer knowledge base at http://developer.motorola.com/techresources/techsupport/.
Here you can search our solution database by product, category, keyword or phrases to quickly find an answer to your question. Additionally, you can submit your questions to our technical support team.

Other developer documentation

Use this developer guide together with other reference material and guides provided by Motorola. Some of those resources are listed here. Motorola is continually adding more information to help our developers. For the latest available developer documentation, see https://developer.motorola.com/docstools/.

User guides

- Motorola SDK User Guide
- Motorola MIDway User Guide

API Device Matrix

The API Device Matrix lists all supported handsets and the applicable APIs for each. You can find the API Device Matrix in the Help documentation inside MOTODEV Studio.

Motorola-proprietary APIs

- Motorola Get URL
- Motorola PIM Enhancement API
- Motorola PIM Enhancement without ToDo, see Motorola PIM Enhancement API
- Motorola Scalable Image APIs
- Motorola Secondary Display API
- Motorola CMCC Enhancements API
- Motorola 3D API
- Motorola Scalable JPG Image (deprecated, replaced by Motorola Scalable Image APIs)
- Motorola Scalable Image Enhancements, see Motorola Scalable Image APIs

NOTE: Some features are dependent on network subscription, SIM card, or service provider, and may not be available in all areas.

Media guides

Additional information about creating media applications can be found in the device-specific media guides at http://developer.motorola.com/docstools/mediaguides/.

JSR specifications

There is a wealth of Java Documentation available. Motorola supports the following APIs and is constantly adding support for additional APIs. For the most current information about your specific handset, check the latest API Matrix available within your SDK or within the Motorola Studio for Java ME. For more information about individual JSRs, go to www.jcp.org.

- JSR 30 - CLDC 1.0 API
- JSR 75 - PIM API and fileConnection API
- JSR 82 - Java™ APIs for Bluetooth™ Wireless Technology

- JSR 118 - Mobile Information Device Profile (MIDP) 2.0 API
- JSR 120 - Wireless Messaging 1.1 API
- JSR 135 - Mobile Media API
- JSR 139 - CLDC 1.1 API
- JSR-172 - Web Services API
- JSR 177 - Security and Trust Services API
- JSR 179 - Location API for J2ME
- JSR 184 - Mobile 3D Graphics API
- JSR-185 - Java Technology for the Wireless Industry API
- JSR 205 - Wireless Messaging 2.0 API
- JSR-226 - Scalable 2D Vector Graphics API

Supported handsets

Included here is a list of the supported Motorola OS handsets, in alphabetic order. Motorola is continually adding new handset and for the latest supported handsets, refer to http://developer.motorola.com/products/handsets/.

Table 2: Alphabetic Listing of Motorola OS Handsets

	Supported Motorola OS Handsets
A	A630, A830, A845
C	C300, C380, C650, C975, C980
E	E1000, E1070, E380, E398, E550, E770
L	L2, L6, L6i
M	MOTOKRZR K1, MOTOKRZR K3, MOTOPEBL U3, MOTOPEBL U6. MOTORAZR maxx V6, MOTORAZR V3 (CLDC 1.0), MOTORAZR V3 (CLDC 1.1), MOTORAZR V3e, MOTORAZR V3i, MOTORAZR V3t, MOTORAZR V3x, MOTORAZR V3xx, MOTORAZR2 V9, MOTORIZR Z3, MOTOROKR E1, MOTOSLVR L7, MOTOSLVR L7i/L7e, MOTOSLVR L9/L72
V	V1050, V1100, V180, V195, V197, V220, V300, V360, V365, V400, V500, V550, V551, V600, V620, V635, V80, V975, V980
W	W490, W510

Chapter 2: Downloading and Managing MIDlets

Methods of downloading

To deploy a MIDlet to a physical Motorola device, use either over-the-air (OTA) downloading (Bluetooth or IrDA) or direct cable (USB) downloading through a PC to the target device. The operator can restrict the MIDlet size.

Method 1–OTA

Using the over-the-air method, you connect—via a wireless network—to a content server, 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.

To download the required JAD (Java Application Descriptor) or JAR (Java Archive) file, use the browser to issue a direct URL request either to the appropriate file or to a Wireless Application Protocol (WAP) page that contains a hyperlink to the target file. In MIDP 2.0, you can download the JAR file directly without first downloading the JAD file. The manifest file contains information about the MIDlet.

The transport mechanism that downloads the file is one of two depending on the support from the network operators WAP Gateway and the size of the file requested.

The MOTODEV Technical Articles section, http://developer.motorola.com/, contains a basic OTA server configuration document, *Browser and OTA Provisioning,* that includes appendices on parameter mapping and a compliancy matrix along with details on how to configure the server and also sample WAP pages.

If there is insufficient space to complete an OTA download, the user can delete MIDlets to free-up space.

The handset uses the GET method to download a MIDlet, and the POST method to send the status code to the server. For a list of status codes, see “Status and Error Codes” on page 123.

The following messages can appear during download:

- If the JAR file size does not match the size specified in the JAD, the handset displays "Failed Invalid File." Upon timeout, the handset goes back to the browser.
- If the MANIFEST file is wrong, the handset displays a transient notice "Failed File Corrupt" and returns to the browser after timeout.
- If the JAD does not contain the mandatory attributes, the handset displays "Failed Invalid File", and returns to the browser after timeout.
- When downloading is done, the handset displays a transient notice "Download Completed" and starts to install the application.

- Upon completing installation, the handset displays “Download complete, launch …” Clicking **Yes** launches the MIDlet. After exiting the MIDlet, the handset returns to the browser. Clicking **No** immediately returns the handset to the browser;

Method 2–Bluetooth

It is possible to install MIDlets by Bluetooth transfer, however because this does not use the .jad file, it is not possible to install MIDlets that use special JAD attributes or that are digitally signed.

To install a MIDlet from its .jar file using Bluetooth:

1. Turn on Bluetooth on both devices, and pair.
2. On the PC, open "My Bluetooth Places", and a window showing the location of the .jar file you wish to install.
3. In the displayed objects, there should be an OBEX object. Do not use the FTP object.
4. Drag and drop the .jar file onto the OBEX object.
5. Accept the installation prompt on the handset.

To debug the installation process, you need to use MIDway and Java Application Loader (JAL) with a USB cable (see below for enabling JAL). The procedure in this case (shortened) is:

1. Pair handset & PC.
2. Turn on JAL (Settings - Java Settings - Java App Loader).
3. Connect USB cable.
4. Start MIDway and connect to appropriate COM port.
5. Drag .jar to OBEX object & install.
6. Run MIDlet. (If needed for log).
7. Save MIDway log.

If you don't need to log the installation, you can connect MIDway after installation to simply take a log of the MIDlet running.

Method 3–IrDA

The Infrared Design Association (IrDA) provides wireless connectivity for devices that would normally use cable connections. IrDA is a point-to-point data transmission standard designed to operate over short distances (up to one meter).

Method 4–Direct cable and Motorola MIDway tool

MIDway, a MOTODEV tool, supports USB cable downloads. For more information about MIDway, see the Solutions database at http:developer.motorola.com/techresources/techsupport/. Select “Find an Answer” and type “MIDway” into the Search Text field. It contains the following information:

- MIDway tool executable
- USB driver for the cable
- Instructions on installation (http://developer.motorola.com/docstools/technicalarticles/Using_MIDway.pdf)
- User Guide for the MIDway tool (http://developer.motorola.com/docstools/technicalarticles/Installing_MIDlets_Using_MIDway.pdf)

In addition to the software, use a USB-A to Mini-USB cable.

If you are using MOTODEV Studio for your development, use the MWay tool instead of MIDway.

The MIDway tool works only with devices that support direct cable Java download. Direct cable Java download is NOT available when purchasing a device from a standard consumer outlet.

To confirm support for the MIDway tool, look at the "Java Tool" menu on the handset to see if a "Java app loader" option is available. If it is not, then contact MOTODEV support for advice on how to obtain an enabled handset.

Motorola provides a MIDway User Guide. In addition, the MOTODEV website contains:

- "Installing Java™ ME MIDlet using MIDway Tool”, which outlines the current version of the tool
- FAQs about the MIDway tool at http:developer.motorola.com.

The USER_AGENT string

Use the USER_AGENT string (also known as the HTTP agent) to identify a handset and render specific content to it, based on the information provided in this string; for example Common Gateway Interface (CGI) on a content server. These strings are located in the connection logs on the content server.

To identify USER_AGENT strings on most Motorola phones, see http://www.wirelessmedia.com/phones/make-list_make-Motorola.html.

The User Agent Profile (UAProf) specification is used to capture functionality and preference information for a handset. This information helps content providers adhere to the appropriate format when creating content for a specific device. Some of the information available in a UAProf file includes model specifications such as screen size, multimedia capabilities and allowable messaging formats.

UAProf information for most Motorola handsets can be found at http://uaprof.motorola.com/

Downloading MIDlets

You can download a MIDlet using either a PC connection or a browser. To download a MIDlet through a PC connection, connect the handset to the PC using IrDA, Bluetooth, or USB. When you successfully connect a PC to your handset, a message appears stating that a connection has been made. Only one connection can be active at a time. The preferred methods of download are OTA or MIDway. Bluetooth is supported on many handsets.

Once connected to the WAP browser, you can search for MIDlets and download them to the handset.

Available memory

A handset initially receives information from the JAD file. The JAD contains the MIDlet-name, version, vendor, MIDlet-Jar-URL, MIDlet-Jar-size, MIDlet-Data-size, and can also contain Mot-Data-Space-Requirements, and Mot-Program-Space-Requirements.

Before downloading a MIDlet, the handset checks for available memory. The Mot-Data-Space-Requirements and Mot-Program-Space-Requirements attributes help the KVM (KJava Virtual Machine) determine whether there is enough memory to download and install the selected MIDlet suite. If there is not enough memory, a message is displayed and the application doesn't download. Upon timeout, the handset once again displays the browser. For information about "Memory Full" and other error codes, see "Status and Error Codes" on page 123.

If an application developer adds the Mot-Data-Space-Requirements and Mot-Program-Space-Requirements attributes to the JAD file, a Motorola handset can determine if enough memory exists on the handset before the MIDlet is downloaded. These attributes may or may not be provided in all MIDlets. Two separate prompts are displayed, depending on whether these attributes are present.

In cases where there is not enough memory to download the application, the user must be given a message to delete existing applications to free additional memory.

For more information, see the technical article, "Using JAD Attributes."

The handset must be able to send and receive at least 30 kilobytes of data using HTTP, between the server and the client in either direction, according to the Over the Air User Initiated Provisioning specification.

Rules

- If the Mot-Data-Space-Requirements and Mot-Program-Space-Requirements attributes are present in the JAD, the message, "Memory Full" is displayed. This value takes into account the memory requirements of the MIDlet and the current memory usage on the handset to tell the user exactly how much memory is required. The memory usage is based in kilobyte units. When this error condition occurs, the download is canceled.
- The label, "Mot-Data-Space-Requirements:", and the value of the data space should be on separate lines. The label, "Mot-Program-Space-Requirements:" and the value of the program space should be on separate lines.

- The “Memory Full” message disappears. A dialog screen with a Help softkey and a Back softkey is displayed instead.
- Clicking **Details** gives the user a detailed Help screen containing information about the memory required to download the MIDlet.
- The Help dialog includes a 'More' right softkey label (for those products in which not all the help data can be displayed on a single screen). This label disappears when the user scrolls to the bottom of the dialog.
- Clicking **Back** returns the user to the original browser page.
- If the Mot-Data-Space-Requirements and Mot-Program-Space-Requirements JAD attributes are not present in the JAD file, the handset cannot determine how much memory to free. Thus, when the message “Memory Full” appears and the user clicks **Details**, the message on the Details screen directs the user to *Games and Apps* to free-up some memory.

Installing MIDlets

After the MIDlet is successfully downloaded, the installation process begins.

Available memory

During installation, the handset may determine that there is insufficient memory to complete the installation. This error can occur whether or not the Mot-Data-Space-Requirements and Mot-Program-Space-Requirements JAD attributes are present. The message “Memory Full” is displayed.

Rules

- When this error occurs, the installation process is canceled.
- The “Memory Full” error disappears. A dialog screen with a Help softkey and a Back softkey is displayed instead.
- Clicking **Details** gives the user a detailed Help screen containing information about the additional memory required to download the MIDlet.
- The Help dialog includes a “More” right softkey label (for those products in which not all the help data can be displayed on a single screen). This label disappears when the user scrolls to the bottom of the dialog.
- Clicking **Back** returns the user to the original browser page.

Managing MIDlets

This section discusses:

- Downloading a JAR File without a JAD
- Upgrading a MIDlet
- Reporting Status on Installing and Deleting

Downloading a JAR file without a JAD

In Motorola's MIDP 2.0 implementation, you can download a JAR file without a JAD. You simply click the JAR file link, the file is downloaded, and the download is confirmed before installation begins. The information presented is obtained from the JAR manifest instead of the JAD.

Upgrading a MIDlet

JSR-118 (MIDP 2.0) rules are 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 Record Management System (RMS) should be preserved, you need to make that decision.

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

If the data cannot be saved, you are warned. If you proceed, the application is downloaded. If you decide to save the data from the current MIDlet, the data is preserved during the upgrade and made available for the new application. In any case, an unsigned MIDlet is not allowed to update a signed MIDlet.

Status report on installing and deleting

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

Chapter 3: MIDP Security Model

Introduction

MIDP 2.0 is a sandbox environment designed to prevent an application (MIDlet) from accessing sensitive functionality. Sensitive functiionality includes APIs for network connections, APIs for read/write access, and APIs for messaging. To gain access to protected and restricted APIs, a MIDlet must be trusted via a digital signature from a signing authority.

This chapter reviews the MIDP 2.0 security environment and Motorola's general security policy with regard to MIDlet signing. It assumes that you are familiar with the MIDP 2.0 (JSR 118, Version 1) specification for Java™ ME and are knowledgeable about public key encryption and digital signatures, including their use in Java certificates.

Procedures and guidelines for obtaining digital signatures for two distinct phases in MIDlet development are described:

- During development, you may want to test a MIDlet on a Motorola handset. If the MIDlet uses a sensitive functionality, you need to obtain a Development Certificate from Motorola.
- After development, sales channels and operators may require the production code to be digitally signed before it can be installed on handsets.

Other benefits to signing a MIDlet for the market are:

- Signing can improve the consumer experience by removing security prompts that would otherwise appear.
- Signing ensures that the distributed MIDlet has not been modified.

The procedures described in this chapter apply to the following Motorola handsets, having a Java environment specified by MIDP 2.0 (JSR 118, Version 1) for Java ME:

- GSM Motorola handsets with the Linux operating system
- 3G and GSM handsets with the Motorola OS (excluding Motorola iDEN handsets)

The MIDP 2.0 security environment

MIDP 2.0 is a sandbox environment designed to limit the ability of a MIDP application (MIDlet) to access sensitive functionality. The limitation is enforced by way of protected and restricted Application Programming Interfaces (APIs) within the Java Virtual Machine.

To create compelling handset applications with rich functionality, a MIDlet must have a path to using the sensitive functionality in a safe manner. This access is accomplished through a system of trust. Namely, the MIDlet must be trusted by the handset and/or the consumer. Once the MIDlet is trusted, the handset

can open a pathway to some or all APIs, allowing the MIDlet to access many additional classes and gain much wider appeal.

The Motorola security model contains two types of API:

- Protected API - A MIDlet can access this API when the consumer grants permission or when the MIDlet is trusted by the handset. The consumer grants permission by responding to prompts to deny or allow access.

  Examples (not available on all Motorola Java ME handsets):

  - Messaging (JSR 120)
  - HTTP/HTTPS
- Restricted API - A MIDlet can access this API only when the MIDlet is trusted by the handset. This trust cannot be overridden by the consumer.

  Examples (not available on all Motorola Java ME handsets):

  - JSR 75 FileConnection (Access to the handset file system)
  - JSR 75 PIM (Access to the user contacts database)
  - JSR 179 Location (Access to Global Positioning System (GPS) subsystem)

MIDP trust

Trust is granted to a MIDlet under the following conditions:

- When it has been digitally signed by a source, known as a Signing Authority, trusted by the handset
- When the consumer has granted the MIDlet permission to access protected (but not restricted) APIs

Signing authority trust

The trust granted by way of digital signing is facilitated through X.509 root certificates embedded in Motorola handsets. A MIDlet that is digitally signed with a signing certificate whose fingerprint matches one of the root certificates is deemed trusted by the handset and will be installed into a protection domain associated with the root certificate. Access to restricted APIs is based on the domain into which a MIDlet is installed.

The level of access can be limited or restricted further by an API access policy implemented on the handset by Motorola or a Network Operator.

Signature authority trust is therefore not binary. It is implemented in a layered approach and must be considered carefully when writing a MIDlet specification that uses restricted APIs. For example, there are implications for developers in selecting a sales channel for their MIDlets. A MIDlet trusted by an unbranded handset (generic Motorola retail handset) might not function identically on an operator-branded handset because the Operator's trust policy may differ from Motorola's general trust policy. This is discussed in more detail in the section "Operator Branding."

Security protection domains provide trust. Motorola's retail (default) implementation complies with the MIDP 2.0 Specification (JSR 118) and has four protection domains as follows:

- Untrusted (unsigned MIDlet)
- Trusted Third Party (TTP)
- Operator
- Manufacturer

Consumer consent trust

The trust granted by consumer consent is facilitated by consumer prompts as defined in the MIDP 2.0 specification (JSR 118). Whenever a MIDlet wishes to access a protected API, the consumer is presented with a menu asking whether to deny or allow the MIDlet to access the API.

In the case of restricted APIs, the MIDlet does not have access and the consumer has no means to override this restriction.

Motorola's general security policy

By default, Motorola handsets trust the following signing authorities:

- Motorola (Manufacturer)
- Motorola (Trusted Third Party)
- Unified Testing Initiative (Java Verified – www.javaverified.com)
- Motorola Operator

Generally, a developer can view the certificates that are present on a device by navigating through to the "Certificate Manager" menu:

Settings -> Security-> Certificate Mgmt->Root Certificate

The resulting list contains Browser SSL Certificates and Java ME Certificates. Example names of Java ME Certificates are:

USIllinois=Motorola Manufacturing or TTP Root Certificates.

USUnified = Unified Testing Initiative Root (Java Verified)

In addition to the signing domains (with which the root certificates are associated), an API access policy controls the level of access that the MIDlet has to the API. This level of access is provided to each domain, and enables control over the level of trust a signing authority (whose root certificate is associated with a domain) is given by the handset.

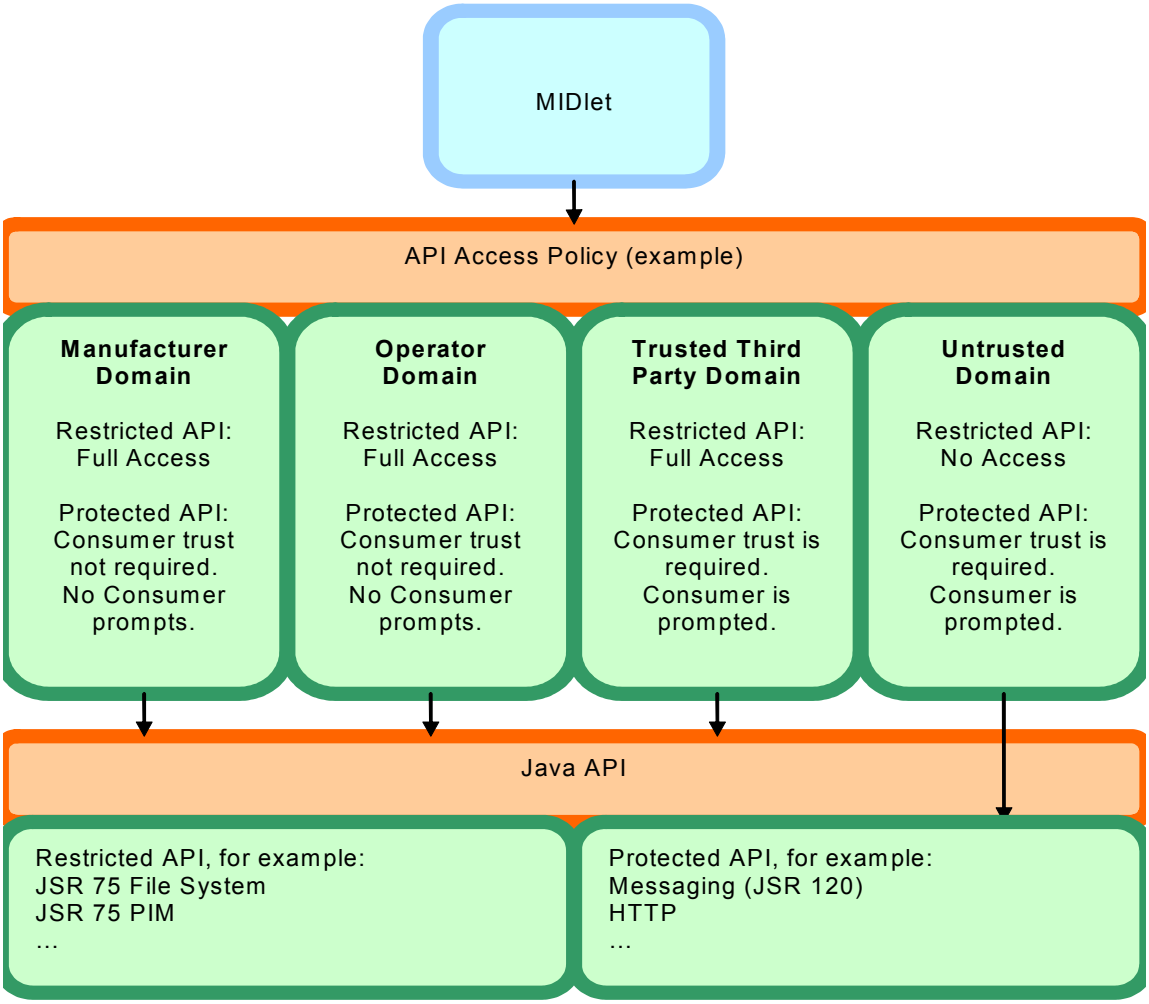

Figure 1: Example Domain and API Policy Model

On a generic Motorola retail handset (and many branded handsets where the operator has not modified the root certificate base or the API access policy), MIDlets digitally signed by the Java Verified Program or Motorola can be installed and run.

Motorola handsets by default (retail/unbranded handsets) provide a trusted signed MIDlet with access to all sensitive APIs, as listed inTable 3. At the time of writing, this list is accurate for the latest Motorola handsets. For an up-to-date, handset-specific API list, refer to “Motorola API Device and Demo Applications Matrix,” in the Software Development Kit associated with the handset.

Table 3: API Access Matrix

Restricted/Sensitive API/Function	Untrusted Access	Trusted Access
Network: • javax.microedition.io.Connector.http • javax.microedition.io.Connector.https • javax.microedition.io.Connector.datagram • javax.microedition.io.Connector.datagramreceiver • javax.microedition.io.Connector.socket • javax.microedition.io.Connector.ssl	Yes (with consumer confirmation prompt)	Yes
Messaging: • javax.microedition.io.Connector.sms • javax.wireless.messaging.sms.send • javax.wireless.messaging.sms.receive • javax.microedition.io.Connector.cbs • javax.wireless.messaging.cbs.receive • javax.microedition.io.Connector.mms • javax.wireless.messaging.mms.send • javax.wireless.messaging.mms.receive	Yes (with consumer confirmation prompt)	Yes
Push Registry: • javax.microedition.io.PushRegistry	Yes (with consumer confirmation prompt)	Yes
Connectivity: • javax.microedition.io.Connector.comm • javax.microedition.io.Connector.bluetooth.client • javax.microedition.io.Connector.bluetooth.server • javax.microedition.io.Connector.obex.client • javax.microedition.io.Connector.obex.server	Yes (with consumer confirmation prompt)	Yes
Multimedia Recording: • javax.microedition.media.control.RecordControl • javax.microedition.media.control.VideoControl.getSnapshot	Yes (with consumer confirmation prompt)	Yes
User Data Read: • javax.microedition.io.Connector.file.read • javax.microedition.pim.ContactList.read • avax.microedition.pim.EventList.read • javax.microedition.pim.ToDoList.read	No	Yes

Table 3: API Access Matrix (Continued)

Restricted/Sensitive API/Function	Untrusted Access	Trusted Access
User Data Write: • javax.microedition.io.Connector.file.write • javax.microedition.pim.ContactList.write • javax.microedition.pim.EventList.write • javax.microedition.pim.ToDoList.write	No	Yes
Location: • javax.microedition.io.Connector.location • javax.microedition.location.LandmarkStore.read • javax.microedition.location.LandmarkStore.write • javax.microedition.location.LandmarkStore.category • javax.microedition.location.LandmarkStore.management • javax.microedition.location.Location • javax.microedition.location.ProximityListener • javax.microedition.location.Orientation	No	Yes

API access — consumer prompts

Consumer prompts may appear to the user whenever a MIDlet attempts to access a protected or restricted API. These prompts guard the user from actions a MIDlet (not trusted by the Motorola or Operator signing authority) might take that costs the user network airtime or allows the MIDlet access to private information such as the user's personal phone book or the file system containing photographs and other private content.

These consumer prompts can be intrusive and break the flow of a MIDlet. Using digital signing (in the correct domain) to trust a MIDlet can remove these prompts and lead to a seamless experience for the consumer.

However, even a digitally signed MIDlet might provide the consumer with an API access prompt. The experience depends on the domain into which the MIDlet will be installed and trusted (for example, the root certificate against which the MIDlet has been digitally signed) and the API access policy implemented on the handset.

Here is a sample scenario:

- A MIDlet is signed against the UTI/Java Verified Signing Domain Certificate. (For information about the Unified Testing Initiative (UTI), see http://www.javaverified.com/about_uti.jsp.
- The MIDlet attempts to access the consumer's contacts database via JSR 75 PIM API.
- The MIDlet is running on a generic Motorola retail handset.

Thie result of this scenario is this consumer prompt:

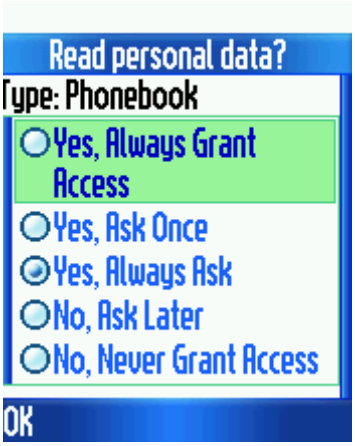

Figure 2: Consumer prompt for JSR 75 PIM API Access

If the consumer selects a positive response, the MIDlet gains access to the restricted API. A negative response renders the restricted API inaccessible either at the time of access or always. Additionally, the future behavior of the MIDlet might be influenced by the consumer's response to the prompts. Using the above example:

- Selecting "Yes, Always Grant Access" grants blanket approval for API access in this instance and all others in the future. This means that the consumer is not prompted again and future executions and consequent API access by the MIDlet are seamless to the consumer.
- Selecting "Yes, Ask Once" grants access to the API in this instance and all others for the duration of the MIDlet execution. This means that the consumer is not prompted again during this session of using the MIDlet. However, when the MIDlet is terminated and restarted, the consumer is prompted on at least the next first access of the API.
- Selecting "Yes, Always Ask" grants a one-shot access to the API. The next API access attempt draws the consumer prompt again.
- Selecting "No, Ask Later" denies access to the API for this instance only. The next API access attempt draws the consumer prompt again.
- Selecting "No, Never Grant Access" blocks access to the API in this instance and all subsequent attempts.

Operator branding

Some operators customize the Java ME security implementation as part of their branding policy. This usually means that a MIDlet must be digitally signed by the Operator as the signing authority for their branded handsets containing their Java root certificate(s).

As Motorola cannot digitally sign MIDlets for the Operator, we recommend that the developer approach the specific operator for digital signing of their MIDlet if the MIDlet is going to be targeted for customers using handsets branded by that specific operator.

These operators/carriers are known to customize the Java ME security policy and root certificate usage:

Amena	Orange	Telefonica
AT&T	Rogers	Ten
H3G (Three)	T-Mobile EMEA/T-Mobile NA	Vodafone/Vodafone France
H3G IL/Partner	LATAM/Movistar	Telenor (Nordic)
Telstra		

NOTE: MIDlets digitally signed by Motorola or Java Verified may not install or run correctly on handsets branded by these operators. Check with your operator for more information.

Identifying installed Java ME root certificates

To find all the root certificates installed on a Motorola handset, look in the following Menu:

Menu > Settings > Security Settings > Certificate Management > Root Certificates

Digital signing and MIDlet development lifecycle

Figure 3 shows a typical development lifecycle for a MIDlet that accesses restricted APIs, and therefore needs to be digitally signed:

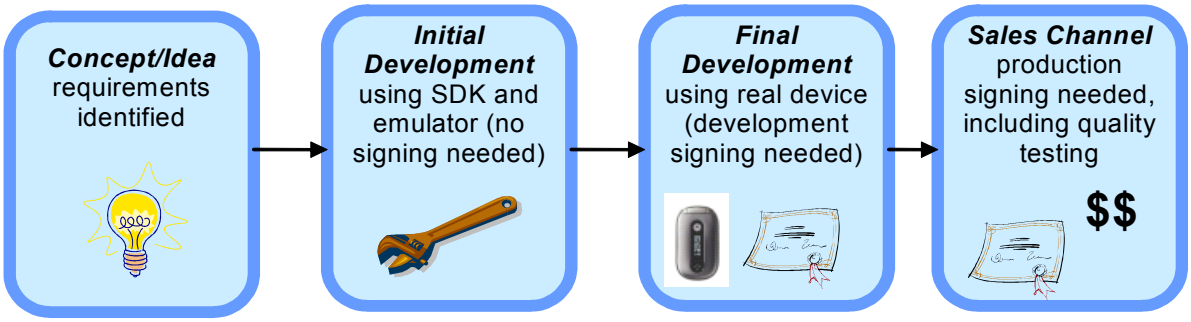

Figure 3: Digital Signing During MIDlet Development

MIDlet development is best done in stages:

1. Concept and specification writing
2. Initial development using a device emulator for testing code
3. Final testing on a real handset

4 Project completion, and acceptance

5 Application delivery to a sales channel

If a MIDlet does not need to access restricted APIs, it doesn't need to be trusted and digital signatures are not required. However, when a MIDlet needs to access restricted APIs, the MIDlet must be trusted. Therefore, it is impossible to test an untrusted MIDlet on a real device because the Security Manager blocks access to restricted APIs.

On-device testing

You can do much of the development of a MIDlet requiring access to restricted APIs using a Motorola emulator. This is because the APIs are deliberately open and available to the MIDlet in the emulated environment. When the MIDlet is near completion, you will want to test the MIDlet on a device to ensure that it will work correctly. However, untrusted MIDlets are blocked from accessing the required API on a handset, and a SecurityException is raised if an untrusted attempt is made.

To facilitate testing on a device, the MIDlet must be trusted by way of a "limited" development signing certificate signature. (More information about Motorola development certificates appear later in this paper, in the section 'Development Certificates".) The limitation usually takes the form of a restriction on the number of devices onto which the MIDlet can be installed or an expiration time/date. It prevents a potentially untested MIDlet from reaching full production status and being released.

Production signing

Whereas a developer certificate enables final testing on a device during development, a trusted signing authority provides final production signing. To obtain production signing, the MIDlet must undergo and pass some form of quality testing by the signing authority.

Once the MIDlet is signed by the trusted signing authority, it is deemed production signed and can be delivered to a sales channel for wide distribution. It is no longer limited to specific devices or an expiration time/date.

So where does a digital signature reside in the MIDlet and how is a digital signature identified? A digital signature consists of two JAD attributes that are placed in the JAD file:

- MIDlet-Jar-RSA-SHA1
- MIDlet-Certificate-1-1

The "MIDlet-Jar-RSA-SHA1" attribute holds the JAR signature. This ensures that the JAR file does not change between the signing authority generation of the signature and installation onto the target device.

The "MIDlet-Certificate-1-1" attribute holds the signature of the signing certificate that matches a root certificate in one of the protection domains.

Additionally, if the MIDlet is to access restricted/sensitive APIs, a MIDlet-Permissions attribute is also required. This attribute must contain all restricted API paths in a comma-separated list as shown in the following example:

```
MIDlet-Permissions:
   javax.microedition.io.Connector.file.read,javax.microedition.io.Connector.file.write
```

These attributes can be placed manually (copy/paste) or via an IDE/Tool into the JAD file. For a list of these attributes, refer to the JSR for the specific API and/or the MIDP 2.0 JSR at http://jcp.org.

Development certificates

Once a MIDlet has been developed as far as possible using the Motorola SDK and emulator, it needs testing on an actual handset to ensure correct functionality and behavior. If the MIDlet accesses sensitive functionality, the MIDlet would have to be trusted to access restricted APIs. This trust is achieved by way of a development certificate.

MIDlets that meet both of the following conditions require a development certificate:

- The MIDlet runs in a CLDC 1.1, MIDP 2.0 Java ME environment
- The MIDlet uses restricted APIs that are otherwise not accessible

MIDlets that do not use the restricted functionality do not require a development certificate.

A development certificate allows the developer to digitally sign the MIDlet in a restrictive way while allowing the MIDlet to access all restricted APIs for testing before final production digital signing.

Bound certificates

Motorola employs a development certificate on CLDC 1.1 products known as a “Bound” certificate. This certificate restricts the developer from digitally signing a MIDlet for mass distribution on production handsets by embedding the processor ID of the handset(s) into the certificate.

A bound certificate restricts the number of handsets onto which the developer can install the development-signed MIDlet.

When a development certificate is created, the Unique Identifier (UID) is embedded into a Motorola development certificate, and thus the certificate is deemed to be “Bound” to one or more handsets. These certificates contain:

- The UID(s) of the developer’s handset(s)
- The Motorola Java root certificate fingerprint
- The developer’s public key

This means that:

- MIDlets signed with the development certificate can only be installed on a handset whose UID matches one of those embedded in the certificate.

- MIDlets signed with the development certificate can only be installed on a handset that has the Motorola Java root certificate embedded.
- The certificate can only be successfully used (to sign a MIDlet) by the developer whose private key matches the public key that was used to create the development certificate.

UID extraction

Motorola handsets internally store a lot of handset-specific information and configurations. Two examples of this type of information are the Internation Mobile Equipment Identity (IMEA) and the Universal Identifier (UID). This data might be needed to perform specific tasks associated with the development of a new Java MIDlet.

One example of such a need is the internal Motorola process to request a certificate to sign a Java ME MIDlet suite. In order to do this, you need the UID of the handset. To access the internal configurations of a handset, it is necessary to connect to the handset via USB. Motorola provides a USB driver package for their handsets, which may be downloaded from the MOTODEV web site.

MOTODEV Studio for Java ME includes a Config Tool. The main purpose of the MOTODEV Config Tool is to provide an easy way for developers to read and write certain specific internal Motorola device configurations. You can extract the device UID using the Config Tool available from MOTODEV Studio on the MOTODEV web site.

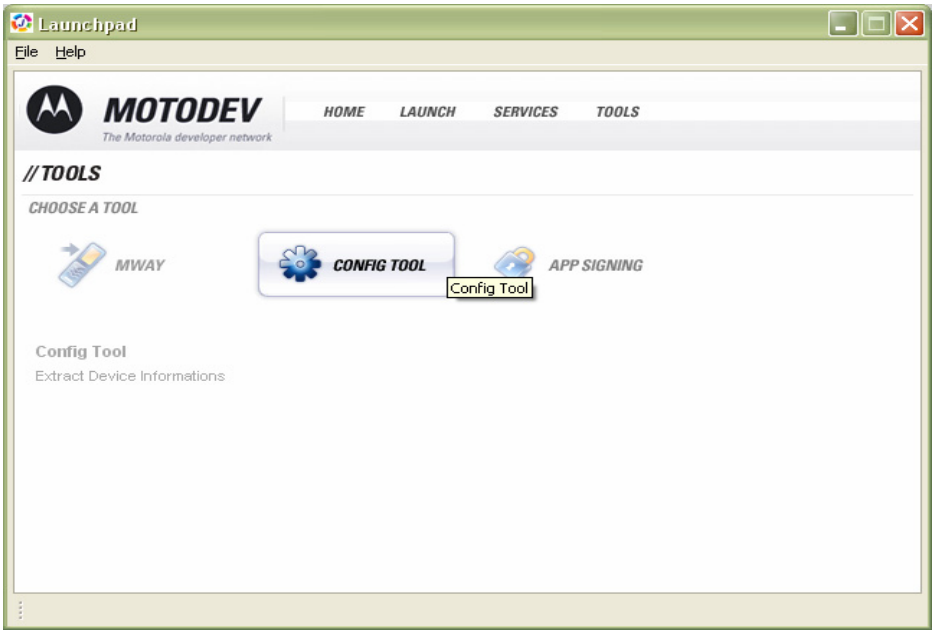

Figure 4: UID Extraction using Motorola Config Tool

If you are not using MOTODEV Studio, you can get the required information by using the UID Extraction Tool

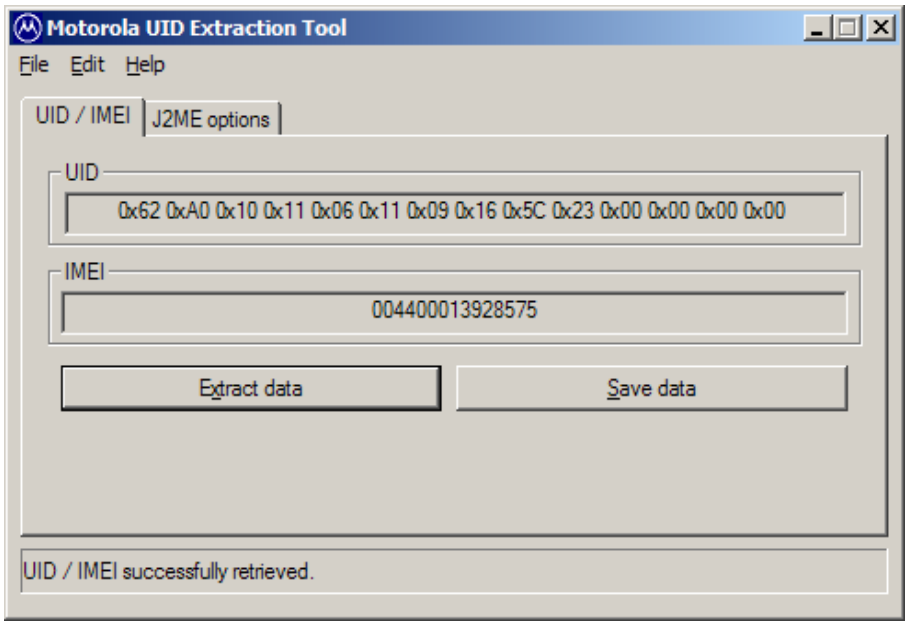

Figure 5: UID extraction using the Motorola UID Extraction Tool

NOTE: A valid UID must be 14 bytes long. Per the example shown, remove all the '0x' prefixes before submitting the UID. If a UID is less than 14 bytes long, a developer must append zeros to the end of the value to make the UID exactly 14 bytes long.

Obtaining a development certificate from Motorola

Step 1: Apply for a development certificate

To request a development certificate, sign up to become a MOTODEV member. If you are already a member, log in to Developer Technical Support (DTS) and complete the *Ask a Question or submit a Bug Report* form as follows:

- **Topic:** "Developer Certificate Request"
- **Category:** "J2ME – MIDP2.0 Enabled Handsets"
- **Subcategory:** Select the applicable handset
- **Subject:** "Request Development Certificate"
- **Question:** Include the reason for your submission as well as any additional information that will assist in processing your request

When the form is complete, click **Submit Question**.

NOTE: A developer must attach a CSR file and a UID file in order for Motorola to process and issue a development certificate. Please carefully review FAQ 940 (using the link that follows) for step-by-step guidelines and information in generating these two required files.

URL: https://motocoder.custhelp.com/cgi-bin/motocoder.cfg/php/enduser/std_adp.php?p_faqid=940&p_created=1156562799

Step 2: Order a bound certificate

Submit a request for technical support through the MOTODEV Technical Support web site in accordance with the following FAQ titled: Bound Certificate User Guide and Request Form.

https://motocoder.custhelp.com/cgi-bin/motocoder.cfg/php/enduser/std_adp.php?p_faqid=940&p_created=1157562799

Title your request “MIDlet Signing: Developer Certificate Request for <your_company_name>.” Ensure that you include all the information that you gathered in the previous step.

Follow the guide to extract UIDs and create a private/public key in the correct format for use with a Motorola bound certificate. You can submit your public key and UIDs in the incident/question created in step 1.

Step 3: Install the bound certificate

MOTODEV processes your request and sends you a bound certificate in about two business days. You can then install the certificate and proceed with signing and MIDlet testing on the bound development handset(s).

Production signing (MIDlet signing)

To place a MIDlet in a sales channel after development and testing, the completed production code (JAR) must first be signed into one of the MIDP 2.0 security domains that control access to restricted APIs. An application is assigned to a domain through the digital signature embedded into the MIDlet JAD file. During MIDlet install, that digital signature is matched to a root certificate on the handset.

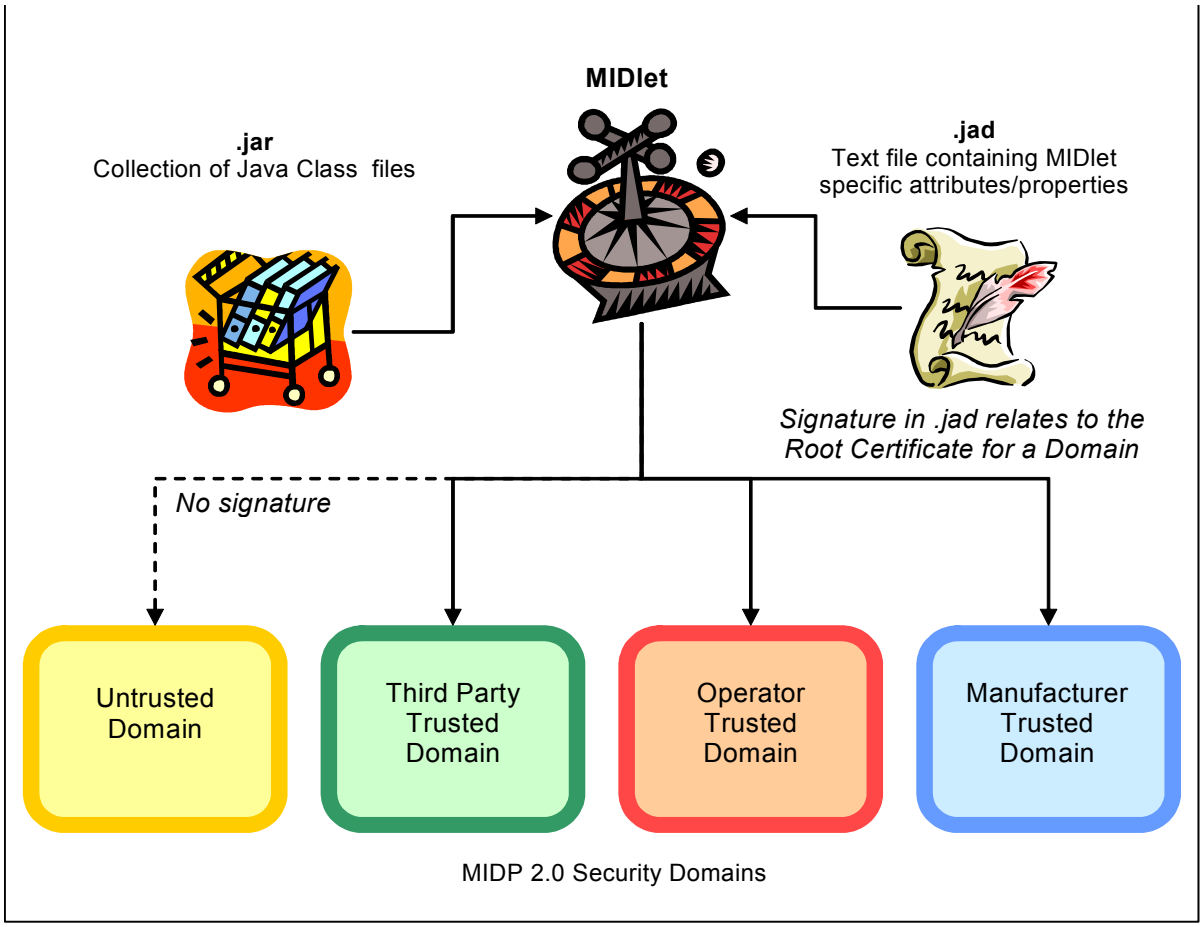

Figure 6: A signature relates a MIDlet to a MIDP 2.0 Security Domain

Choosing a signing authority

Not all MIDlets need to be signed. Depending on the required features as well as the desired consumer experience, developers can choose among three signing authorities:

For the majority of MIDlets, MOTODEV recommends the Java Verified Program. Developers should ascertain that the target handsets for the MIDlet carry the Java Verified or Unified Testing Initiative (UTI) root certificate. Java Verified maintains a list of supported Motorola phones on the Java Verified Web site.

For MIDlets that rely on features unique to an operator or phones that carry only the operator's root certificate, developers must contact the relevant operator.

For special cases where a MIDlet accesses proprietary Motorola features, you may need to apply to Motorola for a manufacturer's signature. Motorola reserves the Manufacturer Trusted Domain for developers with an existing business relationship with Motorola. Because operators customize handsets, you must confirm with the operators that the targeted handsets continue to contain Motorola's features and the Motorola root certificates.

All MIDlets that are targeted for digital signing by Motorola must be fully tested and undergo Motorola's quality assurance process.

The following table, "Comparison of Security Domains," offers an overview of differences among these four security domains.

Table 4: Comparison of MIDP 2.0 Security Domains

	Security Domains			
	Untrusted	**Trusted Third Party**	**Operator**	**Manufacturer**
Signature	Not required	Required	Required	Required
API Access	Limited	Limited	Full	Full
Limited to an Operator	No	No	Yes	No

Production signing authority — summation

Depending on the need for certain features on a handset as well as the desired consumer experience, developers can choose among three signature authorities:

- Java Verified—for signing into the Trusted Third Party domain
- Operator—for signing into the Operator domain
- Motorola—for signing into the Manufacturer domain

Each signing authority has its own testing procedure for the production code and other criteria for issuing signatures.

Motorola production code signing

Motorola reserves its signing authority for MIDlets that reflect Motorola's mission of achieving seamless mobility through iconic design. If your application meets our criteria, we will contact you personally and assign you a Business Sponsor who will guide you through the production code signing process.

Motorola security configuration

On a generic Motorola retail handset (and many branded handsets where the operator has not modified the root certificate base or the API access policy), MIDlets digitally signed by the Java Verified Program or Motorola can be installed and run.

Motorola handsets by default (retail/unbranded handsets) provide a trusted signed MIDlet with access to all sensitive APIs, as listed in the following table. At the time of writing, this list is accurate for the latest

Motorola handsets. For an up-to-date, handset-specific API list, refer to “Motorola API Device and Demo Applications Matrix,” in the Software Development Kit associated with a specific handset.

Table 5: Trusted signed MIDlet access (default values are shown in parentheses)

API	Untrusted	Third-party Trusted	Operator	Manufacturer
DataNetwork javax.microedition.io.Connector.http javax.microedition.io.Connector.https javax.microedition.io.Connector.datagram javax.microedition.io.Connector.datagramreceiver javax.microedition.io.Connector.socket javax.microedition.io.Connector.serversocket javax.microedition.io.Connector.ssl	session (oneshot)	blanket oneshot (session)	Allow	Allow
Messaging javax.microedition.io.Connector.sms javax.wireless.messaging.sms.send javax.wireless.messaging.sms.receive javax.microedition.io.Connector.cbs javax.wireless.messaging.cbs.receive javax.microedition.io.Connector.mms javax.wireless.messaging.mms.send javax.wireless.messaging.mms.receive	(oneshot)	(oneshot)	Allow	Allow
AppAutoStart Javax.microedition.io.PushRegistry	session (oneshot)	blanket (session) oneshot	Allow	Allow
ConnectivityOptions javax.microedition.io.Connector.comm com.vodafone.io.Remotecontrol javax.microedition.io.Connector.bluetooth.client javax.microedition.io.Connector.bluetooth.server javax.microedition.io.Connector.obex.client javax.microedition.io.Connector.obex.servr	blanket (session) oneshot Note: com.vodafone.io is not allowed	blanket (session) Note: com.vodafone.io is not allowed	Allow Note: com.vodafone.io.Remotecontrol is not allowed	Allow

Table 5: Trusted signed MIDlet access (default values are shown in parentheses)

API	Untrusted	Third-party Trusted	Operator	Manufacturer
MultimediaRecording javax.microedition.imedia.control.RecordControl javax.microedition.imedia.control.VideoControl.getSnapshot	session (oneshot)	blanket (session)	Allow	Allow
UserDataReadCapability com.motorola.phonebook.readaccess com.vodafone.midlet.ResidentMIDlet javax.microedition.io.Connector.file.read javax.microedition.pim.contactList.read javax.microedition.pim.EventtList.read com.motorola.smsaccess.readaccess javax.microedition.pim.ToDoList.read	Not allowed	blanket session (oneshot)	Allow Note: com.vodafone.midlet.ResidentMIDlet is not allowed	Allow
UserDataWriteCapability com.motorola.phonebook.writeaccess javax.microedition.io.Connector.file.write javax.microedition.pim.contactList.write javax.microedition.pim.EventList.write com.motorola.smsaccess.writeaccess javax.microedition.pim.ToDoList.write	Not allowed	blanket session (oneshot)	Allow	Allow
Location javax.microedition.io.Connector.location javax.microedition.location.LandmarkStore.read javax.microedition.location.LandmarkStore.write javax.microedition.location.LandmarkStore.category javax.microedition.location.LandmarkStore.management javax.microedition.location.location javax.microedition.location.ProximityListener javax.microedition.location.Orientation	Not allowed	blanket (session) oneshot	Allow	Allow
MotoService com.motorola.io.file.drm.read com.motorola.io.file.drm.write	Not allowed	Not allowed	Not allowed	Allow

Table 5: Trusted signed MIDlet access (default values are shown in parentheses)

API	Untrusted	Third-party Trusted	Operator	Manufacturer
SmartCardCommunications javax.microedition.apdu.sat (see note following table) javax.microedition.apdu.aid	Not allowed	blanket (session) oneshot	Allow	Allow

NOTE: javax.microedition.apdu.sat is not allowed for Trusted Third party nor for Manufacturer.

Summary

- Motorola employs two types of trust in its MIDP security model: Certificate Authority (CA) and consumer consent trust.
- Motorola MIDP 2.0 security is implemented in accordance with the MIDP 2.0 specification using Manufacturer, Operator, Trusted Third Party, and Untrusted protection domains.
- In Motorola's generic retail handsets, all APIs are accessible to a (signing authority) trusted MIDlet installed in any trusted protection domain.
- Consumer prompts may be presented to the consumer depending on the protection domain into which the MIDlet is installed.
- Operator branding may modify the standard (generic retail) Motorola API access policy. Therefore, the developer is advised to investigate the operator's Java security policy before targeting a branded handset for the MIDlet sales channel.
- When signing a MIDlet, you will need additional attributes in the JAD file.
- For testing MIDlets on a handset, the developer needs a development certificate (known as a "bound certificate") for Motorola handsets.

For production signing of MIDlets, Motorola recommends using the Java Verified Program. We recommend the developer ensure that the targeted handset is supported by the Java Verified program and, when the handset is branded, that the operator has not removed the UTI (Java Verified) Java root certificate.

Chapter 4: Network APIs

Network connections

The Motorola implementation of Networking APIs supports the following network connections:

- CommConnection for serial interface
- HTTP connection
- HTTPS connection
- Push registry
- SSL (Secure Socket Layer)
- SocketConnection
- Datagram (UDP or User Datagram Protocol)

Table 6 lists the Network API feature/class support for MIDP 2.0:

Table 6: Network API Feature/Class Support for MIDP

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 of the javax.microedition.io package	READ, WRITE, READ_WRITE
The timeouts parameter for the open() method in the Connector class of the javax.microedition.io package	
HttpConnection interface in the javax.microedition.io package	Supported
HttpsConnection interface in the javax.microedition.io package	Supported
SecureConnection interface in the javax.microedition.io package	Supported
SecurityInfo interface in the javax.microedition.io package	Supported
UDPDDatagramConnection interface in the javax.microedition.io package	Supported
Connector class in the javax.microedition.io.package	Supported
PushRegistry class in the javax.microedition.io package	Supported
CommConnection interface in the javax.microedition.io package	Supported
Dynamic DNS allocation through DHCP	Supported

The following code sample shows the implementation of Socket Connection:Socket Connection.

Code Sample 1: Socket Connection

```
public void makeSocketConnection() {
...

  try {
   // open the connection and i/o streams
   sc = (SocketConnection)Connector.open("socket://www.myserver.com:8080",
     Connector.READ_WRITE, true);
   is = sc.openInputStream();
   os = sc.openOutputStream();
  } catch (IOException io) {
       closeAllStreams();
       System.out.println("Open Failed: " + io.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(); // clean up
        }

  } else {
        // add some code here
  }

}
```

User permission

To add additional network connections, the handset user must explicitly grant permission.

Indicating a connection to the user

When the Java implementation makes additional network connections (the handset is actively interacting with the network), the network icon (a coffee cup) appears on the handset's status bar (Figure 7).

Figure 7: Network Connections Example

Conversely, when the network connection is no longer in use, the network icon disappears from the status bar.

Some handsets support applications that run when the flip is closed. In such situations, the network icon appears on the external display when the application is running in an active network connection.

CommConnection API

The CommConnection API defines a logical serial port connection. This port is part of the underlying operating system. For example, you could configure an IrDA IRCOMM logical serial port. For more information, see http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/io/CommConnection.html.

HTTPS connection

The Motorola implementation supports a HyperText Transfer Protocol Secure (HTTPS) connection on the handset. Additional supported protocols include: Transport Layer Security (TLS) protocol version 1.0, as defined in http://www.ietf.org/rfc/rfc2246.txt; Secure Socket Layer (SSL) protocol version 3.0 as defined in http://wp.netscape.com/eng/ssl3/ssl-toc.html.

Code Sample 2 shows the implementation of HTTPS

Code Sample 2: HTTPS

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

  try {
    hc = (HttpConnection)Connector.open("https://" + url + "/");
  } catch (Exception ex) {
    hc = null;
    System.out.println("Open Failed: " + ex.getMessage());
  }
```

```
if (hc != null) {
  try {
    is = hc.openInputStream();
    byteCounts = 0;
    readLengths = hc.getLength();
    System.out.println("readLengths = " + readLengths);

    if (readLengths == -1) { readLengths = BUFFER_SIZE; }
      int bytes_read = 0;
      int offset = 0;
      int bytes_left = (int)readLengths;
      do {
        bytes_read = is.read(buffer, offset, bytes_left);
        offset += bytes_read;
        bytes_left -= bytes_read;
        byteCounts += bytes_read;
      } while (bytes_read > 0);

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

  } catch (Exception ex) {
      System.out.println("Downloading failed: "+ ex.getMessage());
      numPassed = 0;
  } finally {

     // close input stream
     try {
         is.close();
         is = null;
     } catch (Exception ex) {
       // do something
       System.out.println("Trying to close input stream: " + ex.getMessage() );
     }

     // close https connection
     if (hc != null) {
       try {
         hc.close();
         hc = null;
       } catch (Exception ex) {
         // do something
         System.out.println("Trying to close HTTPS connection: " + ex.getMessage() );

  }
    }
   }
}
```

DNS IP

The Domain Name System (DNS) IP is flexed on or off (per operator requirement). It may or may not be available under Java Settings as read-only or as user-editable. In some instances, it is flexed with an operator-specified IP address.

Network access

Untrusted applications use the normal HttpConnection and HttpsConnection APIs to access web and secure web services. There are no restrictions on web server port numbers through these interfaces. The

implementations augment the protocol so that web servers can identify untrusted applications. The following are implemented:

- The implementation of HttpConnection and HttpsConnection includes a separate User-Agent header with the Product-Token "UNTRUSTED/1.0". User-Agent headers supplied by the application are not deleted.
- The implementation of SocketConnection using TCP sockets throws a `java.lang.SecurityException` when an untrusted MIDlet suite attempts to connect on ports 80 and 8080 (http) and 443 (https).
- The implementation of SecureConnection using TCP sockets throws a `java.lang.SecurityException` when an untrusted MIDlet suites attempts to connect on port 443 (https).
- The implementation of the method DatagramConnection.send throws a `java.lang.SecurityException` when an untrusted MIDlet suite attempts to send datagrams to any of the ports 9200-9203 (WAP Gateway).

The above requirements are applied regardless of the API used to access the network. For example, the javax.microedition.io.Connector.open and javax.microedition.media.Manager.createPlayer methods throw a `java.lang.SecurityException` if access is attempted to these port numbers through a means other than the normal HttpConnection and HttpsConnection APIs

Push registry

The push registry mechanism allows an application to be automatically started. The push registry maintains a list of inbound connections.

Mechanisms for push

Motorola's implementation for push requires the support of the following mechanism:

Short Messages (SMS) push—an SMS with a port number associated with an application used to deliver the push notification. Restricted ports that must not be used are 2805, 2923, 2948, 2949, 5502, 5503, 5508, 5511, 5512, 9200, 9201, 9203, 9207, 49996, 49999.

The JSR-118 specification details the formats for registering SMS.

Push Registry Declaration

The application descriptor file includes information about static connections that the MIDlet suite needs. If all static push declarations in the application descriptor cannot be fulfilled during installation, then the MIDlet suite is not installed. The user is notified of any push registration conflicts. This notification accurately reflects 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 also results in an error and cancellation of the MIDlet installation.

Two types of registration mechanisms are supported.

- Registration during installation through the JAD file entry using a fixed port number
- Dynamic registration using an assigned port number

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

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

Code Sample 3 is an example of a Push Registry implementation.

Code Sample 3: Push Registry

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

public class MyPushTest 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 MyPushTest() {
      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.getString(i));

          registeredConnsCG.delete(i);

          if(registeredConnsCG.size() == 0){

          unregForm.delete(0);

          unregForm.append("No registered connections found.");
                                                                }
                                        }
                                }
            }
          } catch (Exception e) { showMessage("Unexpected " + e.toString() + ": " +
          e.getMessage()); }
        }

      }
    }
}
```

Code Sample 4: WakeUp.java

```
// WakeUp.java

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

Code Sample 5: SMSSend.java

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

public class SmsSend 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 SmsSend() {
      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();
      }
    }

    // inner class?
    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());
        }
      }
    }

// end SmsSend
}
```

Push message delivery

A push message intended for a MIDlet on a handset handles the following interactions:

- MIDlet running while receiving a push message—if the application receiving the push message is currently running, the application consumes the push message without user notification.
- No MIDlet suites running—if no MIDlets are running, the user is notified of the incoming push message and is given the option to run the intended application (Figure 8).

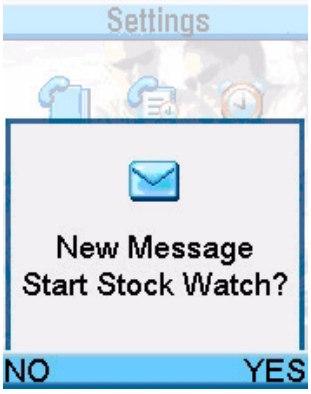

Figure 8: Run Intended Application Query

Table 7: Push Registry Delivery

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 uses 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 is presented with an option to launch the application that had registered for the push message. If the user selects the launch, the current MIDlet is terminated.
Stacked push messages	it is possible for the handset to receive multiple push messages at one time while the user is running a MIDlet. The user is given the option to allow the MIDlets to end and new MIDlets to begin. The user is given the ability to read the messages in a stacked manner (stack of 3 supported), and if not read, the messages are discarded.
No applications registered for push	if there are no applications registered to handle this event, the incoming push message is ignored.

Deleting an application registered for push

If an application registered in the Push Registry is deleted, the corresponding push entry is 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 JSR-118 dealing with MIDP 2.0 and security issues.

Chapter 5: Platform Request API

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

For MIDP 2.0, the `javax.microedition.midlet.MIDlet.platformRequest()` method is called and used when the MIDlet is destroyed.

MIDlet request of a URL that interacts with browser

When a MIDlet suite requests a URL, the browser comes to the foreground and connects to that URL. The user has access to the browser and control over any downloads or network connections. The initiating MIDlet suite continues running in the background. If it cannot, (upon exiting the requesting MIDlet suite) the handset brings the browser to the foreground with the specified URL. If the URL specified refers to a MIDlet suite, JAD, or JAR, the request is treated as a request to install the named package. The user can control the download and installation process, including cancellation. Note that the normal Java installation process is used. For more details, see the JAD Attributes chapter.

MIDlet request of a URL that initiates a voice call

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

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

Chapter 6: RMS API

The Record Management System (RMS) API manages data stored locally on the handset

If a data space requirement is not specified in the MIDlet's JAD attribute (MIDlet_data_space_requirement) or manifest file, 512 KB is the maximum RMS space allowed.

The RMS feature/class support for MIDP 2.0 follows the `javax.microedition.rms` package, as described on the Java web site: http://java.sun.com/javame/reference/apis/jsr037/javax/microedition/rms/package-summary.html. The Motorola implementation supports setting the first record to zero. Motorola also supports:

Interfaces

- RecordComparator
- RecordEnumeration
- RecordFilter
- RecordListener

Classes

- RecordStore

Exceptions

- InvalidRecordIDException
- RecordStoreException
- RecordStoreFullException
- RecordStoreNotFoundException
- RecordStoreNotOpenException

Chapter 7: Gaming API/Multiple Key Press

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, extensive use of native code, hardware acceleration, and device-specific image data formats, as needed.

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

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 by giving the user the ability to press two (2) keys simultaneously so that the corresponding actions of both keys occur simultaneously. For example, simultaneously moving to the right and firing at objects in a game.

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

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

Set 2 — 2, 4, 6, 8, 7

Set 3 — 0, #

Sprite recommended size: 16*16 or 32*32

Chapter 8: iTAP API

The Java™ ME environment has the ability to use iTAP (Intelligent Keypad Text Entry API). iTAP utilizes predictive text technology for easier messaging and note-taking on mobile phones. The use of a predictive entry method is a compelling feature in a MIDlet.

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

Predictive text entry allows a user to simply type in the letters of a word using only one key press per word, and the software provides suggestions for completion. Whereas the TAP method can require as many as four or more key presses to complete the desired word. The use of the iTAP method can greatly decrease text-entry time.

The following Java™ ME text input components 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 is placed into a Form. It is given a piece of text that is used as the initial value.

Refer to Table 8 for iTAP feature/class support for MIDP 2.0:

Table 8: iTAP Feature/Class

Feature/Class	Description
Predictive text	This capability is offered when the constraint is set to ANY.
Text input method	The user can change the text input method during the input process when the constraint is set to ANY (if predictive text is available).
Multi-tap input	Multi-tap input is offered when the constraint on the text input is set to EMAILADDR, PASSWORD, or URL.

Chapter 9: JSR-30 CLDC 1.0

Java ME applications targeting resource-limited devices, such as mobile phones, can benefit from using the Connected Limited Device Configuration (CLDC). On Motorola handsets, the implementation of CLDC 1.0 is based on JSR 30 from the Java Specification Request. Details of the specification are available on the Java Community Process (JCP) web site, http://jcp.org/en/jsr/detail?id=30.

Chapter 10: JSR-75 PDA Optional Packages

PIM API

JSR-75 API is an optional package that provides access to Personal Information Management (PIM) data. The PIM package provides access to personal information—such as Contact, Events, and ToDo lists—that reside natively on devices.

The management of calendars, contact lists, events and alarms, and tasks, are examples of PIM data in cellphone devices.

Details of the specification are available on the Java Community Process (JCP) website http://jcp.org/en/jsr/detail?id=75.

FileConnection API

JSR-75 FileConnection API, provides access to file systems as well as removable storage media, such as a memory card, supported by Motorola devices.

On Motorola devices, the implementation of the FileConnection API is based on JSR-75 from the Java Specification Request. Details of the specification are available on the Java Community Process (JCP) website: http://jcp.org/en/jsr/detail?id=75.

Chapter 11: JSR-82 - Bluetooth API

JSR-82, Bluetooth API, provides wireless, short distance connection between devices for applications such as peer-to-peer networking.

On handsets supporting Bluetooth, Motorola supports both the `javax.bluetooth` package and the `javax.obex` package.

On Motorola devices, the implementation of the Bluetooth API is based on JSR-82 from the Java Specification Request. Details of the specification are available on the Java Community Process (JCP) website: http://jcp.org/en/jsr/detail?id=82.

Chapter 12: JSR-118 MIDP 2.0 Application Testing and Signing

For almost every mobile application, a handset needs sensitive functionality. Sensitive functionalities include network access to a local file system, phonebook, or some other feature accessed through the KVM.

MIDP (Mobile Information Device Profile) 2.0 has a security policy that prevents an application from having access to these functionalities; however, a MIDlet can gain access to a restricted resource if it has a trusted digitial signature from a signing authority.

For more information about permissions, certificates and signing processes for a MIDlet, how Motorola handsets deal with the MIDP 2.0 security policy, and the benefits of having a signed MIDlet, access the MIDlet Testing and Signing section on the MOTODEV portal. For details on Motorola's implementation of MIDP 2.0 security, see Chapter 3 of this guide.

To find out if your handset supports MIDP 2.0 (JSR 118), refer to the latest API Matrix, available on MOTODEV and also in Motorola Studio and the Motorola SDK.

Chapter 13: JSR-120 - WMA

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 handset. This can take the form of chess moves being passed between two players via the WMA.

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

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

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. 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 the address contains a port number, the TP-User-Data of the SMS contains 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 does 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`. For more information refer to sections A.4.0 and A.6.0 of 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; If this is attempted, a `SecurityException` is thrown. 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, review A.3.0 in the JSR-120 specification. A MIDlet uses an SMSC to determine the recipient number.

SMS storing and deleting received messages

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

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 complies 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

Some examples of SMS interaction with a MIDlet are:

- A MIDlet handles an incoming SMS message if the MIDlet is running and registered to receive messages on the port (identifier).
- When a MIDlet that is registered to receive messages on the port number of the incoming message pauses, the user is 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 table lists Messaging features/classes supported in the device.

Table 9: List of Messaging features/classes

Feature/Class	Implementation
Number of MessageConnection instances in the javax.wireless.messaging package	32 maximum
Number of MessageConnection instances in the javax.wireless.messaging package	16
Number of concatenated messages.	30 messages in inbox, each can be concatenated from 3 parts. No limitation on outbox (immediately transmitted)

Code Sample 6 shows the implementation of the JSR-120 Wireless Messaging API

Code Sample 6: JSR-120 Wireless Messaging API MyBinaryMessage

```
import javax.wireless.messaging.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;

public class MyBinaryMessage implements BinaryMessage {

    private BinaryMessage binMsg;
    private MessageConnection connClient;
    private int msgLength = 140;
    private String outAddr = "+17072224444:9532";
    private Random rand = new Random();
    private String myAddress;

    public void makeConnection() {
      try {
        /* Create a connection */
        connClient = (MessageConnection) Connector.open("sms://" + outAddr);

        /* Create a new message object */
        binMsg = (BinaryMessage)connClient.newMessage(MessageConnection.BINARY_MESSAGE);

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

        int num = connClient.numberOfSegments(binMsg);

      }  catch (IOException io) {
          System.out.println(io.getMessage());
      }
    }

      /* Create BINARY of 'size' bytes for BinaryMsg */
      public byte[] createMyBinary(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;
      }


      ...

}
```

Code Sample 7: JSR-120 Wireless Messaging API Sample1.java

```
import javax.microedition.io.*;
import javax.wireless.messaging.*;
```

```
import java.io.*;

public class JSR120Sample1  {

    MessageConnection messageConnection;
    MyBinaryMessage messageToSend, receivedMessage = new MyBinaryMessage();
    JSR120Sample1Listener listener = new JSR120Sample1Listener();


    public void handleMessages() {

      // open connection
      try {
        messageConnection = (MessageConnection)Connector.open("sms://:9532");
      } catch (IOException io) {
         System.out.println( io.getMessage() );
      }

      // create a listener for incoming messages
      listener.run();

      // set payload and address for the message to send
      messageToSend.setAddress("sms://+18473297274:9532");

      // send message (by invoking a send method)

      // set address for received messages
      receivedMessage.setAddress("sms://:9532");

      // receive message (by invoking a receive method)
    }


    // inner class
    class JSR120Sample1Listener implements MessageListener, Runnable {
      private int messages = 0;
      private int result;
      private final int FAIL = 1;

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

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

}
```

Chapter 14: JSR-135 - Mobile Media API.

Network connections

The JSR-135 Mobile Media API feature sets are defined for the following types of media:

- Tone Sequence
- Sampled Audio
- MIDI

When a player is created for a particular type, it follows the guidelines and control types listed in the following sections.

Code Sample 8 is an example of the usage of the JSR-135 Mobile Media API:

Code Sample 8: JSR-135 Mobile Media API

```
import javax.microedition.media.*;


public class MyMP3Player {

    Player player;

    public void createPlayer() {
      // 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());
      }
    }


    public void realizePlayer() {
      // Obtain the information required to acquire the media resources
      try {
        player.realize();
      } catch (MediaException e) {
        System.out.println("FAILED: exception for realize: " + e.toString());
      }
    }


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

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


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

}
```

ToneControl

ToneControl is the interface that enables playback of a user-defined monotonic tone sequence. The JSR-135 Mobile Media API implements the public interface, ToneControl.

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

The available method for ToneControl is:

`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 implements the public interface, VolumeControl. VolumeControl settings are:

Volume Settings	specifies the output volume using an integer value between 0 and 100.
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 `getLevel`. If mute is on, the Player doesn't produce an audio signal. If mute is off, the player produces an audio signal and the volume is restored.

Available methods for `VolumeControl`:

`getLevel(int level)`	Gets the current volume setting.
`isMuted(boolean mute)`	Gets the mute state of the signal associated with this VolumeControl.
`setLevel(int level)`	Sets the volume using a linear point scale with values between 0 and 100.
`setMute(boolean mute)`	Mutes or unmutes the Player associated with this VolumeControl.

StopTimeControl

StopTimeControl allows a specific preset sleep timer for a player. The JSR-135 Mobile Media API implements the public interface StopTimeControl.

Available methods for `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 provides a way to check available/supported content types.

Supported multimedia file types

This section lists media file types (with corresponding Codecs) that are supported in products that are JSR-135 compliant. The common guideline is that all Codecs and file types supported by the handset are accessible through the JSR-135 implementation.

Image Media

Table 10: Image Media

File Type	Codec	Functionality
JPEG	JPEG	Capture

Table 11: Image Media

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

Table 12: Media descriptions

Media	Description
Types	still, audio, video, av
Encodings	jpeg, amr, H.264, mp3, mpgeg4
Container	wav, mps, avi, mov, 3gp
Container extension	.wav, .mp3, .avi. etc.
Mime type	audio/amr...
Download/stream	Playback
Playback/capture	
Print	

Audio media

Table 13: Audio Media

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

Table 14: Audio MIME types

File Type	MIME Type	File Extension
MIDI	audio/midi x-midi mid x-mid sp-midi	.mid, .midi, .xmi
MP3 Audio	audio/mpeg	.mp3
WAV	audio/wav x-wav	.wav, .wave
AMR	audio/amr audio/mp4	.amr
iMelody	audio/imy	.imy

Video media

Table 15: Video Media

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

Feature/class support for JSR-135

The multimedia engine only supports prefetching one sound at a time, but two exceptions exist where two sounds can be prefetched at once. These exceptions are:

- 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.
- 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.

Audio mixing

Must support synchronous mixing of at least two or more sound channels. MIDI+WAV must be supported and MIDI+MP3 is highly desirable.

Media locators

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

RTSP and RTP locators

Realtime Transport Protocol (RTP) is an Internet Protocol (IP) that supports realtime transmission of voice and video. RealTime Streaming Protocol (RTSP) is an application layer protocol used to transmit streaming audio, video, and 3D animation over the Internet. RTP locators must be supported for streaming media on devices supporting real time streaming using RTSP. This support must be available for audio

and video streaming through Manager (for playback media stream). RTP can exist without RTSP, but RTSP cannot exist without RTP.

HTTP locator

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

For example, `Manager.createPlayer("http://webserver/tune.mid").`

File locator

File locators must be supported for playback and capture of media. This is specific to Motorola Java ME implementations supporting file system API and not as per JSR-135. The support is available through Manager and RecordControl implementations.

For example, `Manager.createPlayer("file://motorola/audio/sample.mid").`

Capture Locator

Capture Locator is supported for audio and video devices. The `Manager.createPlayer()` call shall return camera player as a special type of video player. Camera player implements VideoControl and supports taking snapShots using VideoControl.getSnapshot() method.

For example, `Manager.createPlayer("capture://camera").`

Security

Mobile Media API follows the MIDP 2.0 security model. APIs making use of recording functionality need to be protected. Trusted third party and untrusted applications must utilize user permissions. Specific permission settings are detailed below.

Policy

The following table shows security policy, set per operator requirements when the handset is shipped.

Table 16: Security Policy

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

Table 16: Security Policy

Untrusted	**Always Ask**, Ask Once Per App, Never Ask, No Access
Manufacturer	Full Access
Operator	Full Access

Permissions

The following table lists individual permissions in the MultimediaRecord function group.

Table 17: Permissions within Multimedia Record

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

NOTE: The Audio/Media formats are carrier and region dependent and may vary in function and availability.

Basic concepts in OMA DRM

Mobile phone users download ring tones, wallpaper, music, movies and games from service providers everyday. Content downloading is a huge part of the mobile business. DRM (Digital Rights Management) prevents the illegal distribution of content and protects the interest of the content owner.

This chapter introduces the basic concepts and mechanisms in Open Mobile alliance (OMA) DRM 1.0/2.0, and compares the differences between them.

DRM standards in the market

Multiple incompatible DRM standards exist. A brief description of them follows.

OMA DRM

OMA DRM is an open digital rights management standard published by Open Mobile Alliance. Most companies in the mobile industry, including many of the most popular operators and manufactures, take OMA DRM as their DRM standard. Now OMA DRM is the governing DRM standard in mobile industry. Two OMA DRM standards have been released: OMA DRM 1.0 was released in September 2002 and OMA DRM 2.0 published in March 2006.

Microsoft Windows DRM

Windows Media DRM released in March 1999 is a private Digital Rights Management standard for the Windows PC and Windows mobile platform. It is designed to provide secure delivery of audio/video

content over an IP network to a PC or other Windows mobile devices in such a way that the distributor can control how that content is used.

Apple iTunes DRM

Apple DRM, also called Fairplay, is the private digital rights management technology created and used by Apple Inc. Apple DRM is used by the iPod and iTunes Store and plugged into Quicktime. The protected songs purchased from the iTunes Store with iTunes are encoded with Apple DRM. Apple DRM encrypts Advanced Audio Coding (AAC) audio files and prevents users from playing these files on unauthorized computers.

Other Private DRM

Other private DRM techniques and products include IBM's EMMS, Adobe's Content Server and Macrovision's SafeAudio, and so on.

OMA DRM 1.0 model

The OMA DRM 1.0 standard was released in September 2002 and is widely used in mobile devices. It defines three application models, each of which is described in detail in the following section.

- Forward-lock
- Combined Delivery
- Separate Delivery

Forward Lock

Forward Lock is frequently used for ring tones and wallpaper subscription and can effectively prevent illegal copying of files. In Forward Lock mode, the content is packaged and sent to the mobile terminal as a DRM message. The mobile terminal could use the content, but could not forward it to other devices or modify it. In Motorola handsets, the Forward Lock content is not encrypted when it is received or when stored in phone memory. When the .dm file is copied to a PC or memory card, it will be encrypted so as to make sure it cannot be used or transferred from the mobile terminal.

The file extension for a Forward Locked file is .dm, which includes the header and the encoded (but not the encrypted) content in it.

Combined Delivery

Combined Delivery is an extension of Forward Lock. In Combined Delivery mode, the digital rights are packaged with a content object in the DRM message. The user could use the content as defined in the rights object, but could not forward or modify it. The rights object is written in DRMREL (DRM Rights Expression Language) and defines the number of times and length of time that the content can be used thus enabling the preview feature.

The file extension for a Combined Delivery file is also .dm, which includes the header, the Rights Object and encoded content.

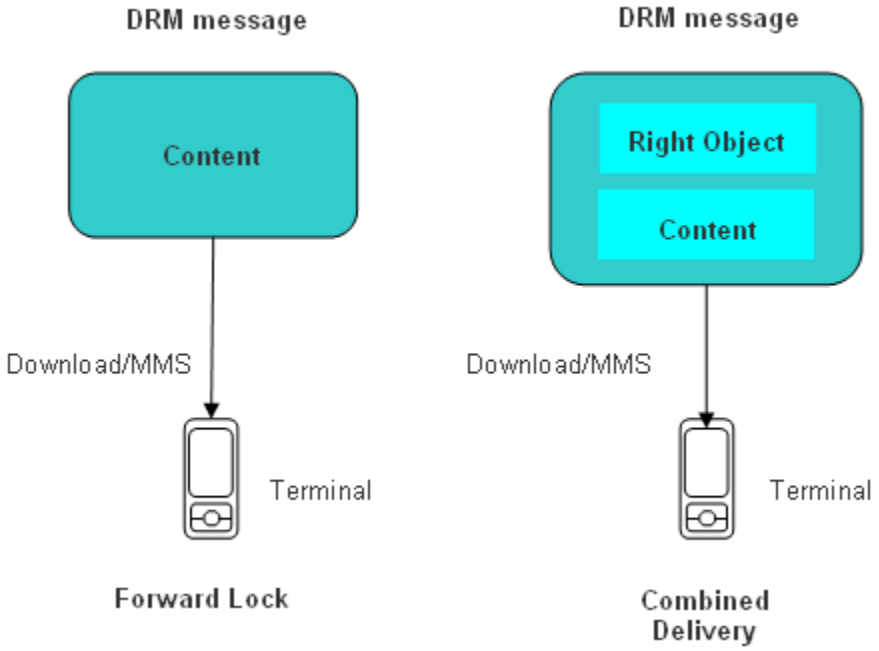

Figure 9: Forward Lock and Combined Delivery

Separate Delivery

In the Separate Delivery mode, the content and rights are packaged and delivered separately. The content is encrypted into DRM Content Format (DCF) using a symmetric cryptograph method and can be transferred in an unsafe way such as Bluetooth, IrDA and via Email. The Rights Object and the Content Encryption Key (CEK) are packaged and transferred in a safe way such as an unconfirmed Wireless Application Protocol (WAP) push. The terminal is allowed to forward the content message but not the rights message.

Superdistribution is a Separate Delivery application which encourages digital content being transferred freely and is typically distributed over public channels. But the content recipient has to contact the retailer to get the Rights object and CEK to use or preview the content.

The encrypted content file type extension is .dcf (DRM Content Format); the right file extension is .dr or .drc.

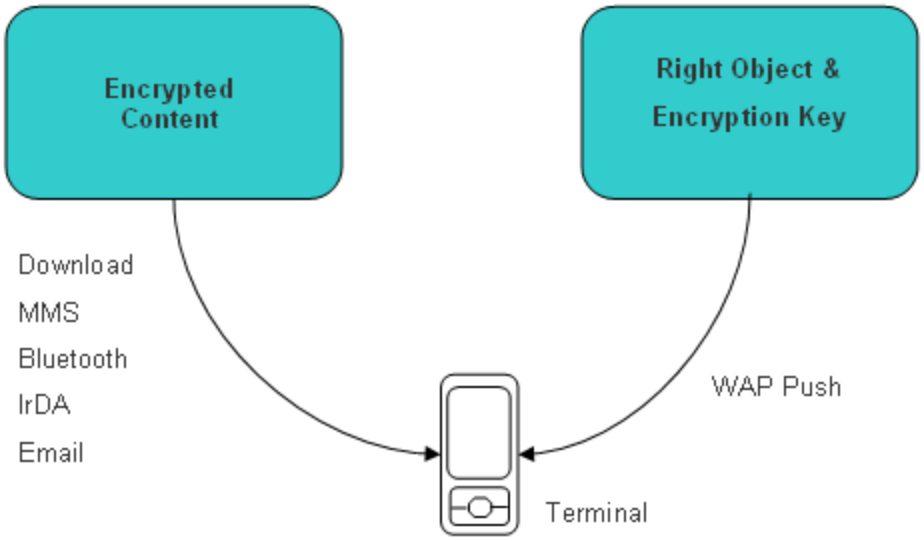

Figure 10: Separate Delivery

Defects in OMA DRM 1.0

The OMA DRM 1.0 model is designed for the mobile industry and is based on the assumption that the mobile terminal is reliable. In the Forward-lock mode and the Combined Delivery mode, the content is not encrypted. In the Separate Delivery mode, the symmetric encryption key is not encrypted. The media content can be stolen if the mobile terminal is hacked or the Right Object message with the CEK is revealed.

OMA DRM 2.0

The OMA DRM 2.0 standard was released in 2006 as an upgrade and extension of version 1.0. It supports many application scenarios like preview, download, Multimedia Messaging Service (MMS), streaming media, super distribution, and unconnected device, making the copyright protection more reliable and flexible.

The OMA DRM 2.0 is composed of four parts:

- Public Key Infrastructure (PKI) security system
- Rights Object Acquisition Protocol (ROAP)
- DRM Content Format (DCF)
- Rights Expression Language (REL)

The Public Key-based Asymmetric Cryptography is used as the basic security mechanism.

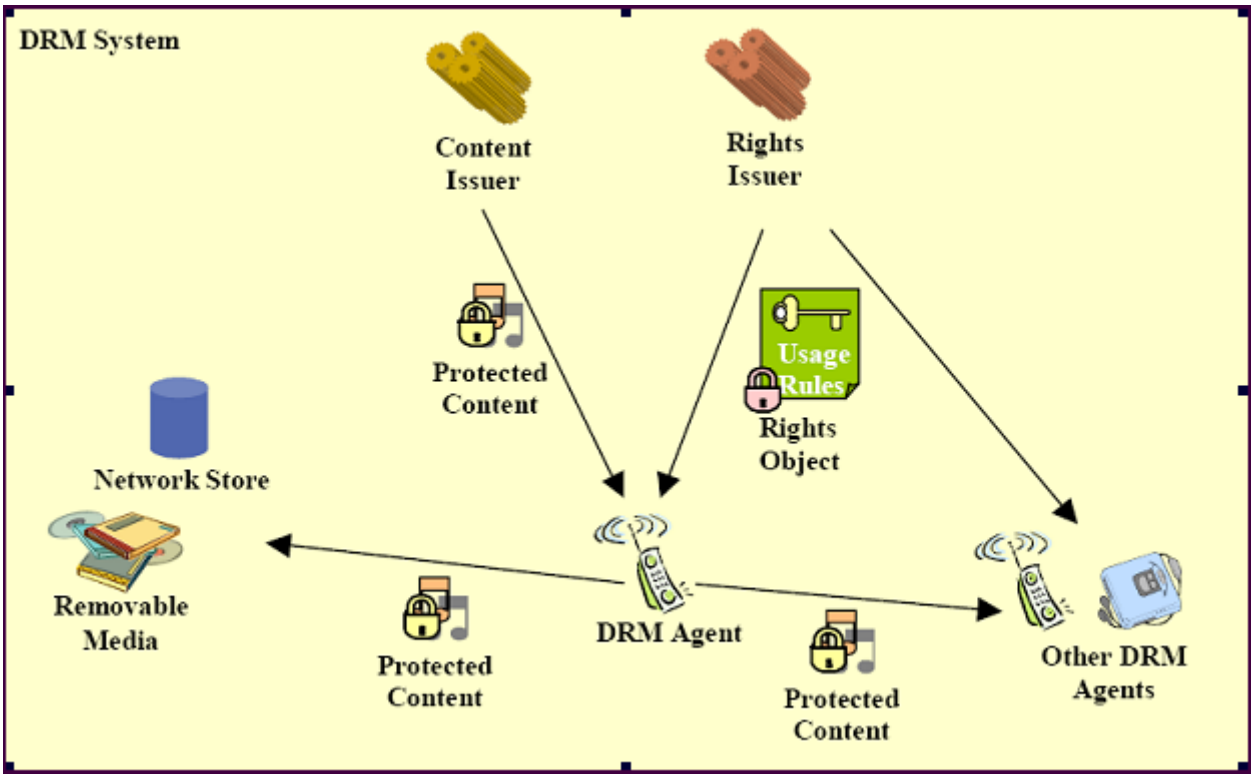

Figure 11: DRM 2.0 architecture diagram from OMA

The diagram above is the DRM System model from OMA Documents. It looks like the Separate Delivery in DRM 1.0 but the Rights Object is signed and passed with the Public Key Infrastructure (PKI) mechanism to ensure security, authenticity and integrity. The DRM Agent is the entity in the device that manages permissions for media objects on the device. With the mobile DRM Agent, devices not connected to a network could use the DRM content.

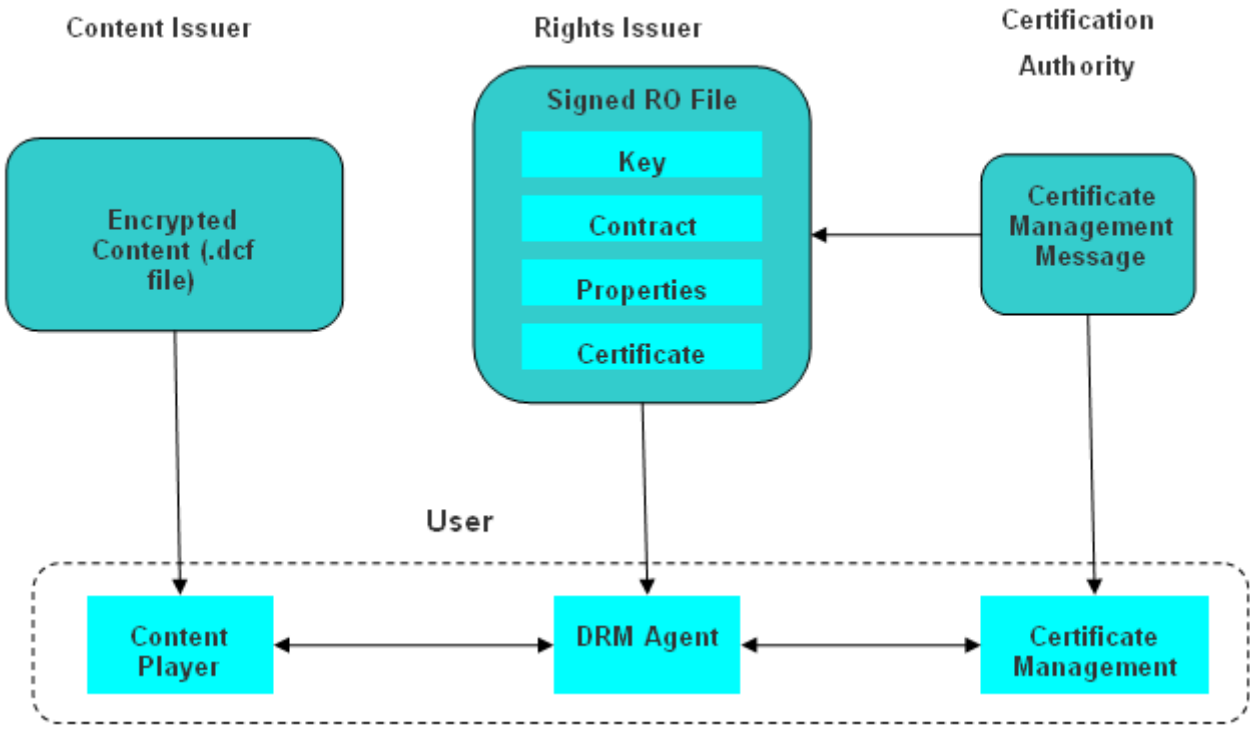

Figure 12: DRM 2.0 content download and use

The diagram above shows how the DRM 2.0 content is downloaded and used.

1. First, the Content Issuer encrypts the original digital content with a symmetric cryptograph algorithm such as AES (Advanced Encryption Standard). The original content is packaged into a DCF-formatted Content Object (CO) and sent to the Content User. The CO does not include the Cryptograph Encryption Key.
2. Second, the DRM agent contacts the Rights Issuer (RI) to get the Right Object (RO) which is generated and managed by RI. In the commercial application this step is fee-based. The CA (Certificate Authority), who issues and verifies certificate, helps the RI and the content user authenticate each other. The RI enciphered the RO with user's public key; then uses the message digest method to get the hash value and signs the RO with RI's private key. After receiving the RO, the user checks the message signature with the RI's public key and decrypts the RO with the user's public key.
3. Third, the user gets the content message digest and symmetric encryption key from RO. Then using the symmetric key to decrypt the CO and comparing the message digest with the content, makes sure it has not been changed. The DRM agent will record the Rights constraint from the RO and control how the content can be used accordingly.

Differences between OMA DRM 1.0 and 2.0

Table 18: Differences between OMA DRM 1.0 and 2.0

	OMA DRM 1.0	OMA DRM 2.0
Application mode	Supports Forward-lock, Combined Delivery, Separate Delivery in content download.	Supports download, MMS, streaming media and many application scenarios like preview, super distribution, unconnected devices, etc.
Domain support	No domain support.	Domain support for a set of devices sharing the same rights.
Mobile phone Support	Most handheld devices support OMA 1.0 including almost all 3G terminals.	Few terminals support it.
Security	Based on the assumption that the terminal is reliable, not secure enough.	Base on the assumption that the terminal is not reliable. The terminal and the server should authenticate each other using the certificates.
Deployment	Easy to deploy	Hard to deploy, CA and certificate system required.

Conclusion

DRM protects the value chain of content download and other value added services. With DRM, the content owner can be properly paid and encouraged to make more valuable content. Motorola supports OMA DRM 1.0 in most handsets and will support OMA DRM 2.0 in the near future.

Chapter 15: JSR-139 - CLDC 1.1

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

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

The support of thread objects to be compliant with J2SE requires the addition of Thread.getName and a few new constructors.

Chapter 16: JSR-172 - Web Services API

JSR-172, Web Services API, is an extension of the Java ME platform to grant access to web services, allowing Java ME devices to be web services clients.

The web services API contains two optional packages:

- Java API for Remote Method Invocations (RMI)
- Java API for XML Processing (JAXP)

On Motorola devices, the implementation of Web Services API is based on JSR-172 from the Java Specification Request. Details of the specification are available on the Java Community Process (JCP) website: http://jcp.org/en/jsr/detail?id=172.

For Motorola-specific system properties, see “System Properties” on page 127.

Chapter 17: JSR-177- SATSA

JSR-177, the Security and Trust Services API (SATSA), provides additional cryptographic security features for the Security Element (SE) in order to enable access to security and trust services and ensure the integrity and confidentiality of the information being transmitted. Motorola OS handsets implement the SATSA-APDU optional package.

SATSA-APDU optional package

The SATSA-Application Protocol Data Units (APDU) package uses an application identifier (AID) to manage the communication from Java™ ME applications to a smart card and vice versa via the APDUConnection interface.

On Motorola devices, the implementation of SATSA is based on JSR-177 from the Java Specification Request. Details of the specification are available on the Java Community Process (JCP) web site: http://jcp.org/en/jsr/detail?id=177.

Chapter 18: JSR-179 Location API

JSR-179 (Location API) is an Optional package used to manipulate the device's geographical data (location, orientation, and physical information).

Details of the specification are available on the Java Community Process (JCP) website http://jcp.org/en/jsr/detail?id=179.

API requirements

Security

The Location API only grants access to trusted applications. If adequate permission is not found, a SecurityException is thrown. Refer to the MIDP 2.0 specification for details http://jcp.org/en/jsr/detail?id=118.

Motorola-specific implementation

Location

- The maximum number of location read requests that can be sent simultaneously from all VMs is 5.
- The default location update interval for location listener is 60 seconds.
- The default location update `maxAge` for location listener is 10 seconds.
- The default location for the `getlocation` method is 30 seconds.

ProximityListeners

- The maximum number of proximity listeners that can be added simultaneously from all VMs is 10.

Landmark

- The maximum number of landmark store categories is 64.
- The maximum number of the landmarks in the landmark store is 256.
- The maximum length of a landmark name is 32 characters (64 bytes).
- The maximum length of a landmark description is 30 characters (60 bytes).
- The maximum size of the landmark store is 312360 bytes.

AddressInfo

- The maximum length of an `AddressInfo` item is 30 characters (60 bytes).

Orientation

- There is no support for any methods in the `Orientation` class.

LandmarkStore

- For the `LandmarkStore` class, neither the create nor the delete `LandmarkStore` methods are supported.

Chapter 19: JSR-184 - Mobile 3D Graphics

This powerful API generates 3D graphics on resource-constrained devices. Even without a GPU this API is capable of drawing sophisticated complex animations and three-dimensional scenes.

Applications for this API include:

- User Interfaces
- Maps Visualization
- Screen Savers
- Games
- Animated Messages

Details of the JSR-184 specification are available on the Java Community Process (JCP) website: http://jcp.org/en/jsr/detail?id=184.

Chapter 20: JSR-185 - JTWI

JTWI (Java Technology for the Wireless Industry) specifies a set of services that enable you to develop highly portable, interoperable Java applications. JTWI reduces API fragmentation and broadens the number of applications for mobile phones.

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

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

For more information, see the JSR-185 specification. In addition, specifications for JSR-120 (Wireless Messaging API 1.1) and JSR-135 (Mobile Media API 1.1) have some content related to JTWI.

Chapter 21: JSR-205 - WMA 2.0

Wireless Messaging API-2.0 (WMA-2.0) is an enhancement of WMA-1.0 (JSR-120) and is supported by many MIDP 2.0 handsets. It enables handsets to send and receive messages of the type, Short Messaging Services (SMS), Cell Broadcast Services (CBS), and Multimedia Messaging Services (MMS), though it is mainly used for sending MMS messages. It handles text, binary, and multipart messages.

For more information about WMA 2.0, see http://jcp.org/en/jsr/detail?id=205, http://java.sun.com/products/wma/index.jsp, and http://developers.sun.com/mobility/midp/articles/wma2/.

Chapter 22: JSR-226 - Scalable 2D Vector Graphics API

The Scalable 2D Vector Graphics API, JSR-226, renders Scalable 2D Vector Graphics (SVG) images. SVG Basic (SVGB), a subset of SVG, is designed for use in mobile devices. SVG Tiny (SVGT) is a further subset of SVGB and is designed for use in cell phones and devices with limited screen size, memory, and bandwidth. The advantage of using vector graphics is that the images scale without distortion to fit the size of the viewing window. Another advantage is that vector graphics file sizes are often smaller and therefore, better suited to resource limited devices.

Applications for this API include any kind of scalable image, zoomable maps, technical illustrations, resizable icons, animated graphics, etc.

On Motorola devices, the implementation of the Scalable 2D Vector Graphics API is based on JSR-226 from Java Specification Request. Details of the specification are available on the Java Community Process (JCP) website: http://jcp.org/en/jsr/detail?id=226.

To find out if your handset supports MIDP 2.0 (JSR 118), refer to the latest API Matrix, available on MOTODEV and also in Motorola Studio and the Motorola SDK.

Chapter 23: Motorola Get URL

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

Flexible URL for downloading functionality

The following rules apply:

- All URLs follow the guidelines outlined in RFC 1738: Uniform Resource Locators (URL). For more information, see http://www.w3.org/addressing/rfc1738.txt.
- URLs are limited to 128 characters.

The Java Application uses the `System.getProperty` method to access the URL. The key for accessing the URL is `com.mot.carrier.URL`. The `System.getProperty` method returns NULL if no URL is present.

Security policy

Only trusted applications have permission to access the Flex system property.

Chapter 24: Motorola PIM Enhancement API

The Motorola PIM Enhancement API implementation provides some extra features that are not supported by the JSR-75 PIM API implementation. It is available in the `com.motorola.pim` package.

The Motorola PIM Enhancement is available in two different versions:

- GSM version
- 3G version

Motorola PIM Enhancement API

Some extra features provided by this API over JSR-75 PIM API are:

- For a given contact, a phone number and email address can be stored simultaneously.
- The name for a contact is stored using the NAME field instead of FORMATTED_NAME field.
- Storing multiple telephone numbers and emails per contact, as compared to a single telephone number and email for a contact.
- Several new data storage fields, for example the `javax.microedition.pim.Contact.EMAIL` field, for adding email information to a contact.
- The 3G version of JSR 75 Enhancement supports the ToDoList; the GSM version does not.

NOTE: For information about handsets that support the PIM API, see the latest version of the Device Matrix, available on the MOTODEV website.

The following table shows the features of the Motorola PIM Enhancement API.I

Table 19: Motorola PIM Enhancement API

Feature	JSR 75 PIM Enhancement
Name Addition to a Contact	Yes
Phone Number Addition to a Contact	Yes
Email Addition to a Contact	Yes
Multiple Email and Phone number Addition	Yes
ToDo List	Yes for 3G; No for GSM.

NOTE: To add the contact ‘s email address, use `javax.microedition.pim.Contact.EMAIL`.

Contact List

Table 20 lists the fields supported by the Contact item of the ContactList for Motorola PIM Enhancement implementation.

- The Motorola extension of the Contact class (`com.motorola.pim.Contact`) is depicted by the term *Motorola Extended*.
- The Java application cannot set the field, DEFAULT_INDEX (default index of the contact). However, if the native phone book sets this field, then the contact has this value.
- The field, LOCATION (integer representation of the UID), supports mailing lists.

Table 20: Contact List - Fields - Motorola PIM Enhancement

Field Description	JSR 75 PIM Enhancement Field	Data Field
Contact Address	ADDR	STRING_ARRAY
Birthdate	BIRTHDAY	DATE
Contact Name	NAME	STRING_ARRAY
Contact Nickname	NICKNAME	STRING
Contact Telephone Number	TELEPHONE	STRING
Contact Email	EMAIL	STRING
Contact Unique ID	UID	INT
Contact Type (Phone, SIM, Mailing List)	CONTACT_TYPE (Motorola Extended)	INT
Contact Photo URL	MOT_PHOTO_URL (Motorola Extended)	STRING
Contact Light ID	Not Supported	N/A
Contact Location	LOCATION (Motorola Extended)	INT
Mailing List Member IDs	MEMBER (Motorola Extended)	INT
Contact Voice Tag	VOICE TAG (Motorola Extended)	INT
Contact Ringtone URL	RINGTONE (Motorola Extended)	STRING
Contact Default Index	DEFAULT_INDEX (Motorola Extended)	INT
Mailing List Type (MESSAGING or MAIL)	MAILING_LIST_TYPE (Motorola Extended)	INT
Contact URL	URL	STRING
Contact Astrological Sign **(3G only)**	ZODIAC (Motorola Extended) **(3G only)**	INT

Table 20: Contact List - Fields - Motorola PIM Enhancement (Continued)

Field Description	JSR 75 PIM Enhancement Field	Data Field
Contact Supplemental information/Notes **(3G only)**	NOTE **(3G only)**	N/A
Content Access Level: CLASS_PRIVATE, CLASS_PUBLIC or CLASS_CONFIDENTIAL	Class **(3G only)**	N/A

NOTE: The following fields are only supported for the JSR PIM Enhancements: DEFAULT_INDEX, MAILING_LIST_TYPE, URL, ZODIAC, NOTE, and CLASS.

Table 21 shows supported attributes for some of the Contact fields.

Table 21: Contact List - Attributes - Motorola PIM Enhancement

Field Label	Attributes in JSR 75 PIM Enhancement
TELEPHONE	PAGER, MOBILE, OTHER, HOME, WORK, FAX, NONE
EMAIL	OTHER, HOME, WORK, NONE
ADDR **(3G only)**	OTHER, HOME, WORK, NONE **(3G only)**

Event list

Table 22 lists various fields supported by an Event item for Motorola PIM Enhancement implementation.

Table 22: Event List - Motorola PIM Enhancement

Field Description	JSR 75 PIM Enhancement Field	Data Type
Relative time for an alarm	ALARM	INT
End time of event	END	DATE
Start time of event	START	DATE
Summary/Subject of event	SUMMARY	STRING
Unique ID for event	UID	STRING

ToDo list

NOTE: This section is only supported by *some* devices. For specifics, refer to the Device API Matrix.

The ToDo database contains entries for tasks that must be executed on determined data and times. Motorola PIM Enhancement 3G is the only implementation type that supports ToDo. Table 23 lists various fields and their data types supported by this implementation.

Table 23: ToDo List - Motorola PIM Enhancement 3G

Field	JSR 75 PIM Enhancement 3G Field	Data Type
Unique ID for a ToDo	UID	STRING
Summary or Subject of the ToDo	SUMMARY	STRING
Priority of the ToDo	PRIORITY	INT
Completion Date for the ToDo	COMPLETION_DATE	DATE
Due date for the ToDo	DUE	DATE
Start date for the ToDo	START (Motorola Extended)	DATE
Status of the ToDo	STATUS (Motorola Extended)	DATE
Motorola Category	MOT_CATEGORY_ID (Motorola Extended)	INT

Chapter 25: Motorola Scalable Image APIs

This proprietary feature provides two Java™ ME APIs, Motorola Scalable Image and Motorola Scalable Image Enhancements, that perform various rescaling operations on arbitrary images.

With this feature a developer can, for example, create thumbnails of pictures taken with the device's camera, or can stretch small icons to fit a rectangular area greater than the icon's original dimensions.

Image scaling modes

Table 24: Available scale modes and their costants

SCALING_MODE_ NON_PROPORTIONAL	This constant sets the non-proportional scaling mode, where the original image will be stretched - width and height will be expanded or reduced to fit the destination area. The scaled image will have horizontal and vertical dimensions equal to that of the destination area. With this operation you can, for example, create a rectangular image from a square image.
SCALING_MODE_ PROPORTIONAL_FIT	This constant sets the proportional scaling mode, where the size of the original image is reduced to a destination area smaller than the original image size. The scaled image will keep both horizontal and vertical dimensions proportional to the original image dimensions. Using this mode you can display a hi-resolution picture taken with the device's camera on its display.
SCALING_MODE_ PROPORTIONAL_EXPAND	This constant sets the proportional scaling mode, where the size of the original image is expanded to a destination area greater than the original image size. The scaled image will keep both horizontal and vertical dimensions proportional to the original image dimensions. Using this mode you can create a "full screen" image from a low-resolution picture taken with the device's camera.
Only in Enhancements API	Only in Enhancements API
SCALING_MODE_PROPORTI ONAL_CUT_CENTERED	This constant sets the proportional scaling mode, where the image is resized and then centered in the target area, being cropped to fit the destination rectangle, if necessary. The scaled image will keep a proportional aspect, although its dimensions after cropping may not be proportional and some details from the original image's borders may also be discarded. With this mode you can create a cropped image from a hi-resolution picture taken with device's camera, removing the borders of the original image and keeping only the middle of the image.

The following diagram demonstrates the modes, with diagram (3) applying only to the enhanced API. The proportional scaling mode minimize is not shown.

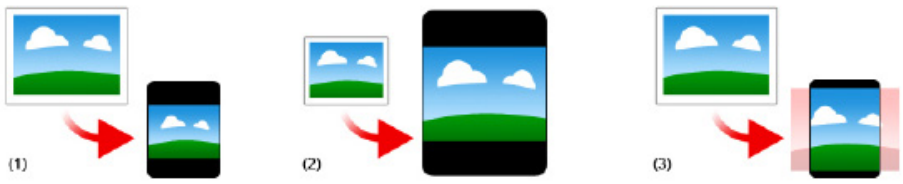

Figure 13: Scaled Image Example

NOTE: In (1) the original image is compressed to fit the destination rectangle.
In (2) the original image is expanded to fit the larger destination rectangle.
In (3) the original image is expanded and cut to fit the destination area (applies to enhancement API only).

Creating a scaled image

Some snippets of code (following), demonstrate how this Motorola feature works. Your application must import the class, `com.motorola.extensions.ScalableImage`.

To create the scaled image, call one of the overloaded versions of the static `CreateImage` method. These overloaded methods are divided into three groups:

Table 25: Overloaded method groups

Raw data arrays	These arrays contain the image data before decoding, including format-specific headers and other non-pixel data. These raw data arrays are internally decoded by the Scalable Image Enhancements API, regardless of the format, assuming that the device can handle this image type.
Input streams	The Scalable Image API can use an arbitrary previously-opened InputStream to fetch the raw image data. These raw image data is internally decoded, as described above.
Image filenames	This API can also load a picture that is inside the device's file system, using a fully qualified filename. It is important to remember that this API does not use the file connection (JSR-75) for file operations, so the protocol suffix "file://" cannot be inserted in the beginning of the filename string.

After the scaled image is created, you can access its internal Image object, using the `getImage` method, and the original image size can be retrieved using `getOrigWidth` and `getOrigHeight` methods.

Code Sample 9 demonstrates how to implement Scalable Image:

Code Sample 9: Scalable Image implementation

```
import javax.microedition.lcdui.*;
import java.io.*;
import com.motorola.extensions.*;

public class ExpandableScalableImage {

    Graphics myGraphics;
    ...

    public void expandImage(String name) {

      try {
        // load the image data
        InputStream myStream = getClass().getResourceAsStream("/small-image.jpg");
        byte[] iconData = new byte[myStream.available()];
        myStream.read(iconData);

        // create a scalable image
        ScalableImage imgFromArray = ScalableImage.createImage(iconData,176, 220, 400,
            500, ScalableImage.SCALING_MODE_PROPORTIONAL_EXPAND);

        // draw the resized image
        myGraphics.drawImage(imgFromArray.getImage(), 0, 0, 0);
      } catch (IOException ioe) {
        ioe.printStackTrace();
      }

    }
```

```
}
```

Code Sample 10: Expanded Non-proportional scalable image example

```
import javax.microedition.lcdui.*;
import java.io.*;
import com.motorola.extensions.*;

public class ExpandedNonPropScalableImage {

    Graphics myGraphics;
    ...

    public void createNonPropImage() {

      try {
        // create a scalable image from a file in phone memory
        String file1 = "/a/mobile/picture/motorola.gif";
        ScalableImage imgFromFile1 = ScalableImage.createImage(file1, 176, 220,
            ScalableImage.SCALING_MODE_PROPORTIONAL_EXPAND);

        // draw the resized image
        myGraphics.drawImage(imgFromFile1.getImage(), 0, 0, 0);

        // create a scalable image from a file in a removable card
        String file2 = "/b/mobile/picture/photo-2007.jpg";
        ScalableImage imgFromFile2 = ScalableImage.createImage(file2, 176, 220,
            ScalableImage.SCALING_MODE_NON_PROPORTIONAL);

        // draw the resized image
        myGraphics.drawImage(imgFromFile2.getImage(), 0, 0, 0);

      } catch (IOException ioe) {
        ioe.printStackTrace();
      }

    }
```

Code Sample 11: Shrunk Scalable Image Example

```
import javax.microedition.lcdui.*;
import java.io.*;
import com.motorola.extensions.*;

public class ShrunkScalableImage {

    Graphics myGraphics;
    ...

    public void shrinkImage() {

      try {
        // create a scalable image from a resource input stream
        InputStream myStream = getClass().getResourceAsStream("/image.png");
        ScalableImage imgFromStream = ScalableImage.createImage(myStream, 44, 55,
            ScalableImage.SCALING_MODE_PROPORTIONAL_FIT);

        // draws the resized image
        myGraphics.drawImage(imgFromStream.getImage(), 0, 0, 0);
```

```
      } catch (IOException ioe) {
        ioe.printStackTrace();
      }

    }

}
```

NOTE: }Code Sample 12 applies to Scalable Image Enhancements only.

Code Sample 12: Scalable Image Enhancement Implementation

```
import javax.microedition.lcdui.*;
import java.io.*;
import com.motorola.extensions.*;

public class ExpandedCutCenteredImage {

    Graphics myGraphics;
    ...

    public void createScalableImage() {

      try {
        // create a ScalableImage from a file in phone memory
        String file1 = "/a/mobile/picture/motorola.gif";
        ScalableImage imgFromFile1 =
        ScalableImage.createImage(file1, 176, 220,
            ScalableImage.SCALING_MODE_PROPORTIONAL_EXPAND);

        // draw the resized image
        myGraphics.drawImage(imgFromFile1.getImage(), 0, 0, 0);

        // create a ScalableImage from a file in a removable card
        String file2 = "/b/mobile/picture/photo-2007.jpg";
        ScalableImage imgFromFile2 =
        ScalableImage.createImage(file2, 176, 220,
        ScalableImage.SCALING_MODE_PROPORTIONAL_CUT_CENTERED);

        // draw the resized image
        myGraphics.drawImage(imgFromFile2.getImage(), 0, 0, 0);

      } catch (IOException ioe) {
        ioe.printStackTrace();
      }
    }

}
```

Chapter 26: Motorola Secondary Display API

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

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

User interface restrictions

The Secondary Display API provides functionality to access the secondary display:

- The secondary display API does not support Screen or Screen's subclasses (Form, TextBox, etc.). Screen and its subclasses support high-level layout and input support.
- The Secondary Display API does not support any input elements like Choice, Item, Text-Field, etc.
- Secondary Display API supports setting Ticker on secondary display.
- The Secondary Display API supports key event processing. Key mappings are supported for Voice and Camera/Smart keys. Extra keys are supported depending on device requirements.
- Only one display, either primary or secondary can have focus at a given time. Primary display is active when flip is open and secondary display is active when flip is closed. Events including key events are delivered to the current active display only.
- The secondary canvas supports full-screen and normal modes. In full-screen mode, the whole secondary display area is available for the MIDlet. In normal mode, the status area is not available for display.
- The Secondary Display API supports all Graphics class functionality.
- Multimedia resources are available for MIDlets running on secondary display, playing audio media and decoding images when the flip is closed.

Flip-open/Flip-close event handling

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

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

The MIDlet receives Flip-Open/Flip-Close events and can take appropriate action based on these events.

Exception handling

For portability purposes, the design of the API allows the developer to handle exceptions related to the instantiation of the secondary display context. Appropriate exceptions are generated for invocation of methods not supported by secondary display.

Push enabled applications

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

Feature interaction

Any incoming call, message, or scheduled native application has priority over a MIDlet running in the secondary display. If a native application requests focus, the running MIDlet is suspended.

Security

The Secondary Display API follows the MIDP security model.

Chapter 27: Motorola CMCC Enhancements API

This China Mobile Communications Corp (CMCC) API consists of two parts:

- User Interface, which implements the scale package
- Phonebook

User interface

Overview

This section describes interface functions applicable to small screens and with simple MIDP operations. The `javax.microedition.lcdui` and `javax.microedition.lcdui.game` packages in the MIDP 2.0 specification define the User Interface. In addition, the `com.cmcc.scale` package is extended according to the requirements of the Java™ service SP in order to implement the function SCALE.

Package

The User Interface part implements the following package, `com.cmcc.scale`.

Interface/class implementation

During SPs provide value-added applications for CMCC, they need stronger system support. So the UI part is extended in Java specification to implement the scale function. The package, `com.cmcc.scale`, implements the interface/class, `ScaleImage`.

ScaleImage

drawScaledRegion

•public static void drawscaledRegion throws ScaleImageException

•IllegalArgumentException, NotSupportScaleReq

This method implements image zoom via copy specified image to target area. The example is in below. The specified method of target area is easy for floating point numbers supporting in the future

`src`	Source Image
`dst`	Destination Graphics
`x_src`	Left/right abscissa of the copied area
`y_src`	Left/right ordinate of the copied area

w_src	The width of the origin area
h_src	The height of the origin area
transform	Transform mode, refer to the definition of javax.microedition.lcdui.Graphics
x_dest	Abscissa of the target anchor
y_dest	Ordinate of the target anchor
w_dst	The width of the destination area
h_dst	The height of the destination area
anchor	Refer to the definition of javax.microedition.lcdui.Graphics
IllegalArgumentException	transform parameter illegal
NotSupportScaleReq	Not support scale requirement defined in the parameter
ScaleImageException	Error in scale operation

The error in the new definition is inherited from java.lang.Exception.

Phonebook

NOTE: This section is only supported by some devices. For specifics, refer to the Device API Matrix.

Overview

This part defines the application program interfaces for accessing the device's phonebook (including information on device and SIM card). These application program interfaces enable third-party developers to conveniently access phonebook information and to better establish point-to-point applications.

Package

The Phonebook part implements the package, com.cmcc.phonebook.

Interface/class implementation

Package javax.wireless.messaging implements the following interface/class:

- PhoneBookEntry
- PhoneBook

Definition of Class

Classes `PhoneBookEntry` and `PhoneBook` are defined In package `com.cmcc. phonebook`.

public class `PhoneBook` extends `java.lang.object`

`PhoneBookEntry` retrieves a record from the address book. Its data structure is identical to the SIM data structure. The variables in the class are public. If the record in the address book of the phone or SIM card does not match these variables, the phone does not process the variables and the variables will be NULL.

Field summary for the field `Java.lang.string`

Table 26: PhoneBookEntry Field Summary:

Name	Contact Person name, default is NULL
Mobile number	Contact Person mobile number, default is NULL
Home number	Contact Person home number, default is NULL
Office phone	Contact Person office number, default is NULL
Email	Contact Person email address, default is NULL
Reserve	Reserved, default is NULL (same as SIM)

`PhoneBook` provides the read record method from the address book. Field summary for `static int`:

Table 27: PhoneBook Field Summary

DEVICE_ALL	Constant, operation for both phone and SIM
DEVICE_PHONE	Constant, operation for phone only
DEVICE_SIM	Constant, operation for SIM card only
SORT_BY_EMAIL	Constant, sorting by email for address book. If no email record is available, sorting by name instead
SORT_BY_Name	Constant, sorting by name for address book
SORT_BY_NOCHANGE	Constant, no sort for address book again

:

Table 28: PhoneBook Method Summary

static void	SetOperateStyle (int sort, int device) — Set address book operation mode, device identifies operate object (SIM or Phone or ALL) default is DEVICE_ALL, sort identifies type of sorting, default is SORT_BY_NAME
static int	SetOperateStyle (int sort, int device) — Return record number in address book
static PhoneBookEntry	getEntry (int index) — Get the record of specified record
static int	findEntryByEmail(java.lang.String email) — get the first record series number suited to email parameter
static int	findEntryByNamel (java.lang.String name) — get the first record series number suited to the first part of name
static int	findEntryByTelNol(java.lang.String tel)get the first record series number suited to phone number

Methods inherited from class `java.lang.Object` are: equals, getClass, hashCode, notify, notifyAll, toString, and wait.

Chapter 28: Motorola 3D API

Recently, more and more phones are starting to support JSR 184 specification (Mobile 3D Graphics API). For many Motorola Java ME devices, such as C975, C980, E1000/E1000R, V975, V980, A780 and E680, the 3D function extends the device's multimedia ability and brings to the customer a new visual experience. It is expected that 3D games or applications based on Mobile 3D Graphics will be significantly increasing with the growing of 3D functions in mobile devices.

Mobile 3D Graphics API introduction

The Mobile 3D Graphics API is an optional package comprising about 250 methods in about 30 classes. This package contains several import classes, as follows:

Object3D class	The most important class because it is the base class of almost all the classes in the package. All the extended classes from Object3D class can be rendered and loaded from m3g file.
World class	The root of the scene graph structure. All 3D objects should be added into the world object. When loading an M3G file, the root 3D object usually is the world object. While rendering a scene, the world object should be passed to Graphics3D object as a rendered object.
Graphics3D class	A singleton 3D graphics context that can be bound to a rendering target. All rendering is done here.
Loader class	A synchronous loader (deserializer) for entire scene graphs, individual branches, and attribute objects. This class can be used to load an M3G file which contains all the 3D objects.

Basic 3D application framework

A 3D image should be rendered on a Canvas or GameCanvas object, so you need to define a class that extends from Canvas or GameCanvas. Besides this, the application usually also needs a timer or thread to show animation or control the movement of the object. A basic 3D canvas implementation follows.

Code Sample 13: M3gCanvas.java

```
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.m3g.Graphics3D;
import javax.microedition.m3g.World;

class M3GCanvas extends GameCanvas implements Runnable {

     Graphics3D g3d;
     World world;

     public M3GCanvas(){
       super(false);
```

```
        setFullScreenMode(true);
        // create and load world and other objects
        Thread t = new Thread(this);
        t.start();
      }

      public void run() {
        Graphics g = getGraphics();
        while (true) {
          // rotate, move or animate object
          try {
            // bind the given Graphics or mutable Image2D
            // as the rendering target of this Graphics3D
            g3d.bindTarget(g);
            // render the world
            g3d.render(world);
          } finally {
            g3d.releaseTarget();
          }
          flushGraphics();
          try {
            Thread.sleep(100);
          } catch (Exception e) {
          }
        }
      }

      ...
}
```

Adding a thread sleep statement here (see following code), frees up CPU resources and allows other threads to run.

Code Sample 14: Thread sleep statement

```
try {
Thread.sleep(100);
}catch (Exception e) {
}
```

The following code is a 3D MIDlet implementation. It creates a 3D canvas in startApp() method.

Code Sample 15: 3D MIDlet implementation

```
import javax.microedition.m3g.*;

public class CreateTetrahedron {

    private void createTetrahedron() {

      Mesh tetrahedron;
      World world;

      // the vertices used by the tetrahedron
      short []POINTS = new short[] {0, 2, 0,  // point 0, top
                                                    0, 0, 1, // point 1
                                    -1, 0, -1, // point 2
                                    1, 0, -1 // point 3
         };
```

```
        // the points sequence
        int []INDICES = new int[] {3, 0, 1,
                                                    0, 1, 2,
                                                    1, 2, 3,
                                                    2, 3, 0
        };

        // the color for each point
        byte []COLORS = new byte[] { 127, 127, 0,
                                                                127, 0, 0, // R
                                                                0, 127, 0, // G
                                                                0, 0, 127 // B
        };

        // the length of each sequence in the indices array
        // the tetrahedron is built by four triangles
        int []LENGTH = new int[] {3, 3, 3, 3};
        VertexArray POSITION_ARRAY, COLOR_ARRAY;
        IndexBuffer INDEX_BUFFER;

        // create a VertexArray to be used by the VertexBuffer
        POSITION_ARRAY = new VertexArray(POINTS.length / 3, 3, 2);
        POSITION_ARRAY.set(0, POINTS.length / 3, POINTS);
        COLOR_ARRAY = new VertexArray(COLORS.length / 3, 3, 1);
        COLOR_ARRAY.set(0, COLORS.length / 3, COLORS);
        INDEX_BUFFER = new TriangleStripArray(INDICES, LENGTH);

        // the VertexBuffer holds references to VertexArrays that
        // contain the positions, colors, normals, and texture coordinates
        // for a set of vertices
        VertexBuffer vertexBuffer = new VertexBuffer();
        vertexBuffer.setPositions(POSITION_ARRAY, 1.0f, null);
        vertexBuffer.setColors(COLOR_ARRAY);

        // create the 3D object defined as a polygonal surface
        tetrahedron = new Mesh(vertexBuffer, INDEX_BUFFER, null);

        // set the appearance of the mesh object
        Appearance appearance = new Appearance();
        PolygonMode polygonMode = new PolygonMode();
        polygonMode.setPerspectiveCorrectionEnable(true);

        // specify that both faces of a polygon are to be drawn
        polygonMode.setCulling(PolygonMode.CULL_NONE);

        // specify smooth shading
        polygonMode.setShading(PolygonMode.SHADE_SMOOTH);
        polygonMode.setTwoSidedLightingEnable(true);
        appearance.setPolygonMode(polygonMode);

        // set the appearance of the 3D object
        tetrahedron.setAppearance(0, appearance);

        // move the tetrahedron into the screen
        tetrahedron.setTranslation(0.0f, -1.0f, -3.0f);
        world = new World();
        world.addChild(tetrahedron);
    }

}
```

Loading a 3D object

The previous code shows a basic 3D application framework without showing how to create a 3D object. There are two ways to acquire a 3D object: from a data array or from an M3G file. The M3G file format is defined in the JSR 184 specification and is provided as a compact and standardized way of populating a scene graph. The following code shows how to create a world object from an M3G file

Code Sample 16: Creating a world object from an M3G file

```
public void loadFile() {
    try {
      // Load a m3g file, returns all root object3d object.
      Object3D[] roots = Loader.load("mytest.m3g");
      // Usually, the world is the first root node loaded.
      myWorld = (World)roots[0];
    } catch(Exception e) {
      e.printStackTrace();
    }
}
```

Loading 3D content from an M3G file is an easy way to program and get 3D content. It can be used to load a complex scene. However, you must use a third-party tool to obtain an M3G file. For some simple applications, it is not necessary to purchase such a tool before programming. Instead of loading 3D content from an M3G file, the object data can be stored in arrays. Using this method, the 3D object is created manually inside the program. The following method creates a colored tetrahedron.

Code Sample 17: Creatig a colored tetrahedron

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.m3g.*;
import M3GCanvas;

public class TetrahedronDemo extends MIDlet implements CommandListener {

    private Command cmdExit;
    private Display d;
    private M3GCanvas m3gCanvas;
    private World myWorld;

    public TetrahedronDemo(){
      d = Display.getDisplay(this);
      cmdExit = new Command("Exit", Command.EXIT, 0);
      m3gCanvas = new M3GCanvas();
      m3gCanvas.addCommand(cmdExit);
      m3gCanvas.setCommandListener(this);
    }

    public void startApp() {
      d.setCurrent(m3gCanvas);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable d){
      notifyDestroyed();
    }
```

```
}
```

The code looks quite complex, but in fact you can re-use many statements. To create another 3D object, you just need to change vertices, indices, colors (if needed), and norm (if necessary) arrays, and a little code. The previous code creates a tetrahedron object as shown in Figure 14.

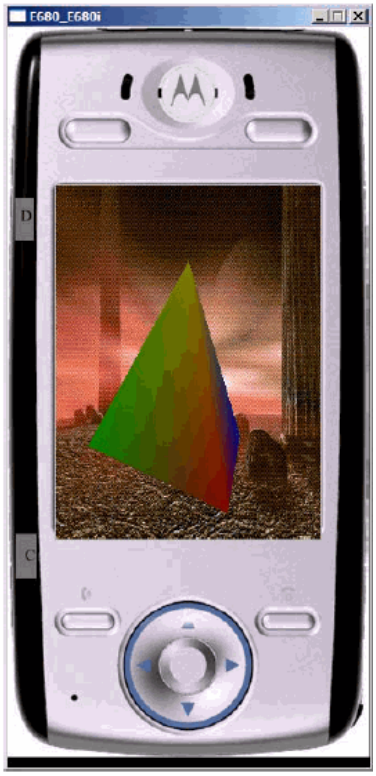

Figure 14: Tetrahedron object

Loading from an array method can create a simple object or a complex scene. To convert a complex scene into a data array, you might need a converter tool. Unlike converting M3G file format, many free tools are available to download. We recommend that you convert a 3D scene file into obj file format and then convert the obj file into a data array. Java 3D API supports obj file format by providing ObjectFile class, which can load obj file into a scene object.

Except for JSR 184 API, some Motorola phones support Motorola 3D API. The two sets of API are not the same, so please pay close attention to device capabilities when choosing a device. For details of which handsets support JSR 184 API and which support Motorola 3D API, see the API Matrix in either the SDK or the MOTODEV Studio documentation.

Appendix A: Key Mapping

The following table identifies key names and corresponding Java assignments. Java does NOT process any other keys..

Table 29: Key Mapping

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.TestBox with PHONENUMBER constraint set)
Center Select	Select

Table 29: Key Mapping

Key	Assignment
End	Handled according to Motorola specification. Pause/End/Resume/Background menu invoked.

Appendix B: JAD Attributes

JAD/manifest attribute implementations

The JAR manifest defines attributes that the Application Manager Software (AMS) uses to identify and install the MIDlet suite. These attributes may or may not be found in the application descriptor.

The Application Manager Software uses the application descriptor in conjunction with the JAR manifest, to manage the MIDlet. The application descriptor is also used:

- By the MIDlet, for configuration specific attributes.
- To allow the Application Manager Software on the handset to verify that the MIDlet is suited to the handset before loading the JAR file.
- To allow 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 (JAD) attributes as outlined in JSR-118. Table 30 lists all MIDlet attributes, descriptions, and locations in the JAD and/or JAR manifest that are supported in the Motorola implementation. Please note that the MIDlet is not installed if the MIDlet-Data-Size is greater than 512k.

Table 30:MIDlet Attributes, Descriptions, JAD, and JAR Manifest

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlet 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 further information describing the MIDlet suite.	Yes	No
MIDlet-<n>	The name, icon, and class of the nth MIDlet in the JAR file. The name identifies 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 is loaded.		Yes
MIDlet-Jar-Size	The number of bytes in the JAR file.		Yes

Table 30:MIDlet Attributes, Descriptions, JAD, and JAR Manifest (Continued)

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Data-Size	The minimum number of bytes of persistent data required by the MIDlet.	Yes	Yes
MicroEdition-Profile	The Java™ ME profiles required. If any of the profiles are not implemented, the installation fails.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR manifest.
MicroEdition-Configuration	The Java™ ME Configuration required, that is, CLDC.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR manifest.
MIDlet-Permissions	Zero or more permissions that are 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

Appendix C: Status and Error Codes

The following status codes and messages 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

Notification

When the MIDlet file size exceeds the maximum value set, a notice informs the user that the MIDlet file download has been aborted.

When the MIDlet file size exceeds the maximum value set and the download is aborted, the following notification is sent to the server: “901 Insufficient Memory.”

Downloading MIDLets

Table 31: Actions and Results

Action	Result
Browser connection interrupted/ended	If the browser connection is interrupted/ended during the download/installation process, the device is unable to send the HTTP POST with the MIDlet-Install Notify attribute. In this case, the MIDlet is deleted to ensure that the user does not get a free MIDlet. This can occur when a phone call is accepted and terminated during installation, because then the browser is not in the state necessary to return the MIDlet Install Notify attribute.
Installation completion	Upon completing installation, the handset displays a transient notice 'Installed to Games and Apps'.
Timeout	Upon timeout, the handset goes back to Browser.
Failed file Corrupt	During Installation, if the MANIFEST file is wrong, the handset displays a transient notice 'Failed File Corrupt'. Upon timeout, the handset goes back to browser display.
Failed Invalid File	If the JAD does not contain mandatory attributes, a “Failed Invalid File” message appears.
Handset flip closed	During the installation process, if the handset’s flip is closed, the installation process continues and the handset does not return to the idle display. When the flip is opened, the 'Installing...' dialog should appear on the screen and should be dynamic.
Timeout	Upon timeout, the handset goes back to browser display.
Voice behavior	During download and installation, voice record, voice commands, voice shortcuts, and volume control are not supported, but incoming calls and SMS messages can be received.

Error logs

Table 32 shows the error logs associated with downloading MIDlets.

Table 32: Error logs

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

Table 32: Error logs (Continued)

Error Dialog	Scenario	Possible Cause	Install-Notify
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 supersedes 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.	910
Application Error	MIDlet Execution	Authorization failure during MIDlet execution: Incorrect MIDlet.	

Messages displayed after download

Table 33: Description of error messages

Message	Description
Download Failed	If an error, such as a loss of service, occurs during download, then the transient notice 'Download Failed' must be displayed. Upon timeout, the handset goes back to an idle state.
Download Cancelled	A downloading application can be cancelled by pressing the END key. The transient notice, 'Download Cancelled,' is displayed. Upon timeout, handset goes back to Browser. When the download is cancelled, the handset cleans up all files, including any partial JAR files and temporary files created during the download process.
Failed Invalid File	This message is displayed if JAR -file size does not match the specified size. Upon timeout, the handset goes back to Browser.

Message	Description
Download Completed	When downloading is done, the handset displays a transient notice “Download Completed.” The handset then starts to install the application. After an application is successfully downloaded, a status message must be sent back to the network server. This allows for charging of the downloaded application. Charging is per the Over the Air (OTA) User Initiated Provisioning (UIP) specification. The status of an install is reported by means of an HTTP POST request to the URL contained in the MIDlet-Install-Notify attribute. The only protocol that MUST be supported is 'http://'. During installation, if the MANIFEST file is wrong, the handset displays the transient notice “Failed File corrupt.”

Appendix D: System Properties

Java.lang implementation

Motorola's implementation for the java.lang.System.getProperty method supports additional system properties to those specified in JSR 118.

The additional system properties are:

`CellID`	The device's current Cell ID is returned during implementation.
`batterylevel`	The application's current battery level is returned, during implementation, as a percentage of full charge
`IMEI`	International Mobile Equipment Identity. The device's IMEI number is returned during implementation.
`MSISDN`	Mobile Station Integrated Services Digital Network. The device's MSISDN of the device is returned during implementation.

The IMEI and MSISDN properties are not available for unsigned MIDlets. For more information on this class, go to http://java.sun.com/j2se/1.4.2/docs/api/java/lang/System.html. The following code sample shows the java.lang implementation.

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

The following information is provided from the java-tips.org web site and can be seen in its entirety at http://www.java-tips.org/java-me-tips/midp/how-to-retrieve-system-properties-in-a-midlet.html.

This Java ME tip illustrates the retrieval of system properties in a MIDlet. MIDlets have direct access to all four of the standard system properties defined by the CLDC specification.

Code Sample 18: Hello world program

```
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

/*
 * A Hello, World program in Java ME MIDP, JSR 118.
 * The class must be public so the device
 * application management software can instantiate it.
 *
 */
public class HelloWorld extends MIDlet {
    public HelloWorld() {}

    public void startApp() {
      // create a widget from a subclass of Displayable
      Form form = new Form("Hello World");
```

```
      // add a string to the form
      String msg = "My first MIDlet!";
      form.append(msg);

      // display the form
      Display display = Display.getDisplay(this);
      display.setCurrent(form);
      printSystemProperties();
    }

    /*
     * Display the values of standard system properties.
     *
     */
      protected void printSystemProperties() {
        String conf;
        String profiles;
        String platform;
        String encoding;
        String locale;

        conf = System.getProperty("microedition.configuration");
        System.out.println(conf);

        profiles = System.getProperty("microedition.profiles");
        System.out.println(profiles);

        platform = System.getProperty("microedition.platform");
        System.out.println(platform);

        encoding = System.getProperty("microedition.encoding");
        System.out.println(encoding);

        locale = System.getProperty("microedition.locale");
        System.out.println(locale);
        System.out.println();
      }


    protected void pauseApp() {
      notifyPaused();
    }


    protected void destroyApp(boolean flag) {
      notifyDestroyed();
    }
}
```

The following table is from the Sun Developer Network (SDN) web page, http://developers.sun.com/mobility/midp/questions/properties/index.html.

This table lists the defined system properties, drawing them from JSRs that are in the public review, final ballot, or final state, as defined in the Java Community Process (JCP):

Java ME defined system properties

Table 34: Java ME system properties (from the Sun Developer Network)

JSR	Property Name	Default Value[a]
30	microedition.platform	null
	microedition.encoding	ISO8859_1
	microedition.configuration	CLDC-1.0
	microedition.profiles	null
37	microedition.locale	null
	microedition.profiles	MIDP-1.0
75	microedition.io.file.FileConnection.version	1.0
	file.separator	(impl-dep)
	microedition.pim.version	1.0
118	microedition.locale	null
	microedition.profiles	MIDP-2.0
	microedition.commports	(impl-dep)
	microedition.hostname	(impl-dep)
120	wireless.messaging.sms.smsc	(impl-dep)
139	microedition.platform	(impl-dep)
	microedition.encoding	ISO8859-1
	microedition.configuration	CLDC-1.1
	microedition.profiles	(impl-dep)
177	microedition.smartcardslots	(impl-dep)
179	microedition.location.version	1.0
180	microedition.sip.version	1.0
184	microedition.m3g.version	1.0
185	microedition.jtwi.version	1.0
195	microedition.locale	(impl-dep)
	microedition.profiles	IMP-1.0
205	wireless.messaging.sms.smsc	(impl-dep)
205	wireless.messaging.mms.mmsc	(impl-dep)
211	CHAPI-Version	1.0

a. (impl-dep) indicates that the default value is implementation-dependent.

Motorola getsystemProperty() keys for Motorola OS devices

The table that follows contains the Motorola getsystemProperty() keys. However, not all properties are available on all Motorola OS handsets. In addition, new properties are added from time to time. Check the MOTODEV website to get the most current information and the API Matrix to get device specifications.

Table 35: Motorola getsystemProperty() keys and their corresponding values

Key	Value
MIDP	
microedition.timezone	Current timezone
microedition.configuration	<Not implemented>
microedition.platform	<Not implemented>
microedition.locale	Locale: <language code>-<country code>
microedition.encoding	<Not implemented>
microedition.profiles	MIDP version and optional VSCL version.
microedition.hostname	Local address to which the socket is bound.
microedition.commports	Discover available comm ports
commports.maxbaudrate	Maximum baud rate of comm ports. For P2K device is 115200
Device	
device.software.version	Device software version
device.flex.version	Device flex version
device.model	Device model ID
batterylevel	Current battery level. Battery values are the following: 0, 1, 2, and 3, based on the battery level.
IMSI	International Mobile Subscriber Identity Code
default.timezone	Current time zone information from the network
language.direction	“0” if left-to-right, otherwise “1”
com.mot.network.airplanemode	Status of Airplane Mode.
JSR75	
microedition.io.file.FileConnection.version	Version of the Java APIs for File Connection. For this version it will be set to "1.0".
microedition.pim.version	Version of the Java APIs for PIM. For this version it will be set to "1.0".
file.separator	File separator: '/'
JSR135	
microedition.media.version	Version of the Java APIs for Multimedia. For this version it will be set to "1.1".
supports.mixing	Sound mixing is supported
supports.audio.capture	Audio capture is supported

Table 35: Motorola getsystemProperty() keys and their corresponding values (Continued)

Key	Value
supports.video.capture	Video capture is supported
supports.recording	Video recording is supported
audio.encodings	Supported audio encodings (e.g., encoding=audio/amr encoding=audio/amr-wb)
video.encodings	Supported video encodings
video.snapshot.encodings	Supported video snapshots (e.g., encoding=jpeg encoding=image/jpeg)
MAType	
GPRSState	
JSR82	
bluetooth.api.version	Version of the Java APIs for Bluetooth wireless technology that is supported. For this version it will be set to "1.0".
bluetooth.l2cap.receiveMTU.max	The maximum ReceiveMTU size in bytes supported in L2CAP. The string will be in Base 10 digits, e.g., "672". This value is product dependent. The maximum value is 64 Kb.
bluetooth.connected.devices.max	Maximum number of connected devices supported (includes parked devices). The string will be in Base10 digits. This value is product dependent.
bluetooth.connected.inquiry	Is inquiry allowed during a connection? Valid values are either "true" or "false". This value is product dependent.
bluetooth.connected.page	Is paging allowed during a connection? Valid values are either "true" or "false". This value is product dependent.
bluetooth.connected.inquiry.scan	Is inquiry scanning allowed during connection? Valid values are either "true" or "false". This value is product dependent.
bluetooth.connected.page.scan	Is page scanning allowed during connection? Valid values are either "true" or "false". This value is product dependent.
bluetooth.master.switch	Is master/slave switch allowed? Valid values are either "true" or "false". This value is product dependent.
bluetooth.sd.trans.max	Maximum number of concurrent service discovery transactions. The string will be in Base10 digits. This value is product dependent.
bluetooth.sd.attr.retrievable.max	Maximum number of service attributes to be retrieved per service record. The string will be in Base10 digits. This value is product dependent.
JSR120	
wireless.messaging.sms.smsc	SMS Message Center (SMSC) address
JSR205	
wireless.messaging.sms.mmsc	MMS Message Center (MMSC) address
JSR185	
microedition.jtwi.version	Version of the JTWI that is supported. For this version it is set to "1.0".
JSR177	
microedition.smartcardslots	Smartcard slots

Table 35: Motorola getsystemProperty() keys and their corresponding values (Continued)

Key	Value
JSR184	
microedition.m3g.version	Version of the Java APIs for Mobile 3G. For this version, it is set to "1.0" or absent
VSCL	
vscl.device.backlight	
vscl.device.blink	
vscl.system.wakeupmode	
vscl.system.silentmode	
vscl.system.javasettingvolume	
vscl.system.javasettingvibration	