



MOTODEV

The Motorola developer network



Java ME Developer Guide for Motorola OS

1.3

Developer Guide

Copyright © 2009, Motorola, Inc. All rights reserved.

This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

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

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

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

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

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

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

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

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

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

If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc.

MOTOROLA and the Stylized M Logo are registered in the US Patent & Trademark Office. Java and all other Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the US and other countries. Linux® is the registered trademark of Linus Torvalds in the US and other countries. All other product and service names are the property of their respective owners.

Java ME Developer Guide for Motorola OS

Version 1.3

January 2009

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

Motorola, Inc.

<http://www.motorola.com>

Contents

Chapter 1	Overview	1
	Purpose and audience	1
	Developer tools	1
	MOTODEV Studio for Java ME	1
	MOTODEV SDK for Java ME	1
	Additional resources	2
	Technical articles	2
	Developer knowledge base	5
	Other developer documentation	5
	JSR specifications	6
	Supported handsets	7
Chapter 2	Downloading and Managing MIDlets	9
	Methods of downloading	9
	Method 1—OTA	9
	Method 2—Bluetooth	10
	Method 3—IrDA	10
	Method 4—Direct cable and Motorola MIDway tool	11
	The USER_AGENT string	11
	Available memory	12
	Rules	12
	Installing MIDlets	13
	Downloading a JAR file without a JAD	14
	Upgrading a MIDlet	14
	Status report on installing and deleting	14
Chapter 3	MIDlet Signing	15
	The MIDP 2.0 security environment	16
	MIDP trust	16
	Motorola's general security policy	17
	API access	19
	Consumer prompts	20
	Operator branding	21
	Identifying installed Java ME root certificates	22

Digital signing and MIDlet development lifecycle	22
On-device testing	23
Production signing	23
Development certificates	24
Bound certificates	24
Unique Identifier	25
UID extraction	25
MIDlet production signing	27
Choosing a signing authority	28
Production signing authority summary	29
Motorola's security configuration	30
Signing a MIDlet with a bound certificate	32
Prerequisites	33
Installation steps	33
Adding UIDs to an existing bound certificate	44
Summary	44
Chapter 4 <i>Motorola 3D API</i>	45
Mobile 3D Graphics API introduction	45
Basic 3D application framework	45
Loading a 3D object	48
Chapter 5 <i>Motorola iTAP API</i>	51
Chapter 6 <i>JSR-118 MIDP 2.0 Application Testing and Signing</i>	53
Platform Request API	53
MIDlet request of a URL that interacts with browser	54
MIDlet request of a URL that initiates a voice call	54
RMS API	54
Interfaces	54
Classes	55
Exceptions	55
Multiple key press gaming support	55
Network connections	55
User permission	57
Indicating a connection to the user	57
CommConnection API	58
HTTPS connection	58

Network access	59
Push registry	59
Mechanisms for push	60
Push Registry Declaration	60
Push message delivery	66
Deleting an application registered for push	67
Security for push registry	67
Command class type priority, influence on placement	67
Chapter 7 JSR-120 - WMA	69
SMS client mode and server mode connection	69
SMS port numbers	70
SMS storing and deleting received messages	70
SMS message types	70
SMS message structure	70
SMS notification	71
Chapter 8 JSR-135 - Mobile Media API	75
Network connections	75
ToneControl	76
VolumeControl	76
StopTimeControl	77
Manager class	77
Supported multimedia file types	77
Image Media	78
Audio media	79
Video media	80
Feature/class support for JSR-135	80
Audio mixing	80
Media locators	80
RTSP and RTP locators	80
HTTP locator	81
File locator	81
Capture Locator	81
Security	81
Policy	81
Permissions	82
Basic concepts in OMA DRM	82
DRM standards in the market	82

Differences between OMA DRM 1.0 and 2.0	88
Chapter 9 JSR-185 - JTWI	89
Appendix A Key Mapping	91
Appendix B JAD Attributes	93
JAD/manifest attribute implementations	93
Custom Motorola JAD attributes - *need to verify*	94
Appendix C Status and Error Codes	97
Notification	97
Downloading MIDlets	98
Error logs	98
Messages displayed after download	99
Appendix D System Properties	101
Java.lang implementation	101
Java ME defined system properties	103
Motorola getSystemProperty() keys for Motorola OS devices	104

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 supported 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 OS 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 an integrated 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 Software Development Kit (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 Integrated Development Environments (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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Remote Control of Audio/Video Using Bluetooth • Using HTTP and HTTPS on Motorola MIDP 2.0 Handsets • Using Serial Connections on Motorola Java ME Handsets
Developer Tools	<ul style="list-style-type: none"> • Motorola's MWay Tool V1.0 • 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	<ul style="list-style-type: none"> • Provisioning with the Open Mobile Alliance Device Management Platform • Creating Device Configurations
DRM	Introduction of Basic Concepts in OMA DRM
Games	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Using the Push Registry in MIDP 2.0 • Changing JAR Files Manually • Deploying and Debugging MIDlets • 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 • 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	<ul style="list-style-type: none"> • Changing JAR Files Manually • MIDlet Lifecycle on Motorola Handsets • Using Push Registry on Motorola Handsets • Using RMS on Motorola Java-Enabled Handsets • Motorola Custom Attributes in JAD Files
Java ME: Language Topics	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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
Location	<ul style="list-style-type: none"> • Introduction to the Java ME Location API
Messaging	<ul style="list-style-type: none"> • Sending and Receiving SMS Messages on UIQ Handsets • 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	<ul style="list-style-type: none"> • Remote Control of Audio/Video Using Bluetooth • JSR-135: The Mobile Media API • Smart Tools for Smartphones - MOTODEV Studio for UIQ • 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
Porting	<ul style="list-style-type: none"> • Porting MIDlets on Motorola Handsets • Porting Java ME Applications from the T720i to the V300, V400, V500 and V600 • Porting a MIDlet from the i95cl to the T720
Optimizing	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • MIDlet Signing • Using Crypto APIs for Secure Communications • Password Based Encryption in Java ME • Proper Speed and Heading Calculation Using Location Services • How to Set Up Your SSL Connection in Linux Devices
Testing and Debugging	<ul style="list-style-type: none"> • Deploying and Debugging MIDlets • Debugging MIDlets on the MOTOSLVR L7 • Using KDWP to Debug MIDlets Running on Motorola Handsets
UIQ	<ul style="list-style-type: none"> • Smart Tools for Smartphones - MOTODEV Studio for UIQ • UIQ/Symbian Development • Implementing Key and Pointer Events via Java ME MIDlets on Motorola Symbian Handsets
Windows Mobile	<ul style="list-style-type: none"> • Programming the Motorola Q Windows Mobile Smartphone • Developing a GPS Application for the MOTO Q9

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

Motorola-proprietary APIs

- [Motorola Get URL from Flex](#)
- [Motorola PIM Enhancement API](#)
- Motorola Bluetooth Remote Control API
- Motorola Digital Rights Management
- Motorola FunLights V2
- Motorola PIM Enhancement without ToDo, see [Motorola PIM Enhancement API](#)
- [Motorola Scalable Image APIs](#)
- [Motorola Secondary Display API](#)
- Motorola Micro3D version 2 API
- Motorola Touch Sensor Music Player Buttons
- Motorola PhoneCall API
- Motorola Short Messages Access API
- Motorola Location API
- Motorola Location API for non-GPS enabled handsets

- Motorola ModeShift Technology
- Motorola FastScroll Navigation Wheel
- 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 - Connected Limited Device Configuration 1.0 API](#)
- [JSR 75 - Personal Information Management 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 - Connected Limited Device Configuration 1.1 API](#)
- [JSR-172 - Web Services API](#)
- [JSR 177 - Security and Trust Services API \(APDU, Crypto, PKI\)](#)
- [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-211 Content Handler](#)
- [JSR-226 - Scalable 2D Vector Graphics API](#)

- [JSR-234 - Advanced Multimedia Supplements API \(Camera, Image processing, Tuner\)](#)
- [JSR-238 - Mobile Internationalization](#)
- [JSR-239 - Java Binding for the OpenGL ES](#)
- [JSR-248 - Mobile Service Architecture](#)

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	MOTO Z9/Z9n, MOTOACTV W6, 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, V950, 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 97.

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 97.

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: MIDlet Signing

Application code signing is primarily to ensure that the application (MIDlet) comes from a trusted source, and possibly, to allow access to protected APIs. Generally an application does not need to be signed in order to install. However, in order to eliminate the message “This application is from an unknown publisher,” the application needs to be signed. Additionally, carriers may restrict access to some APIs which would then require the application to be signed in order for it to be installed.

Application certification is a good way to have an independent entity test and verify the behavior of an application. Application certification involves running the application through a number of test cases to verify correct behavior of the application, that it does not leak memory or threads, and does not disrupt the call handling and data handling capability of the handset.

MIDP 2.0 is a sandbox environment designed to prevent an application from accessing sensitive functionality. Sensitive functionality includes APIs requiring network connections, read/write access, and messaging. To gain access to protected and restricted APIs, a MIDlet must be trusted via a digital signature from a signing authority. The MIDP 2.0 security framework protects the handset and the user from actions that could be damaging to the handset or costly to the user. This framework uses Java Root Certificates assigned to security domains to apply the correct secure access restrictions.

This article reviews the MIDP 2.0 security environment and Motorola’s general security policy with regard to MIDlet signing. Also included are the procedural steps for signing your MIDlet. We assume 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.

There are procedures and guidelines for obtaining digital signatures for two distinct phases in MIDlet development: during development and after development.

During development you may want to test a MIDlet on a Motorola handset. If the MIDlet uses 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.

Some 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 and provides traceability to the publisher/owner of the MIDlet.

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

The MIDP 2.0 sandbox environment is 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 (JVM).

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 APIs:

Protected APIs - A MIDlet can access this type of 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 APIs - A MIDlet can access this type of 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 the 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 trusted by the handset, known as a Signing Authority
- 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 is 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 by 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 in the same way on an operator-branded handset because the operator's trust policy may differ from Motorola's general trust policy. See "Operator branding" on page 21."

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 wants access to a protected API, the consumer is presented with a menu asking whether to deny or allow the MIDlet access to 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 security certificates that are present on a handset by navigating through to the "Certificate Manager" menu:

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

The resulting list contains Browser Secure Socket Layer (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.

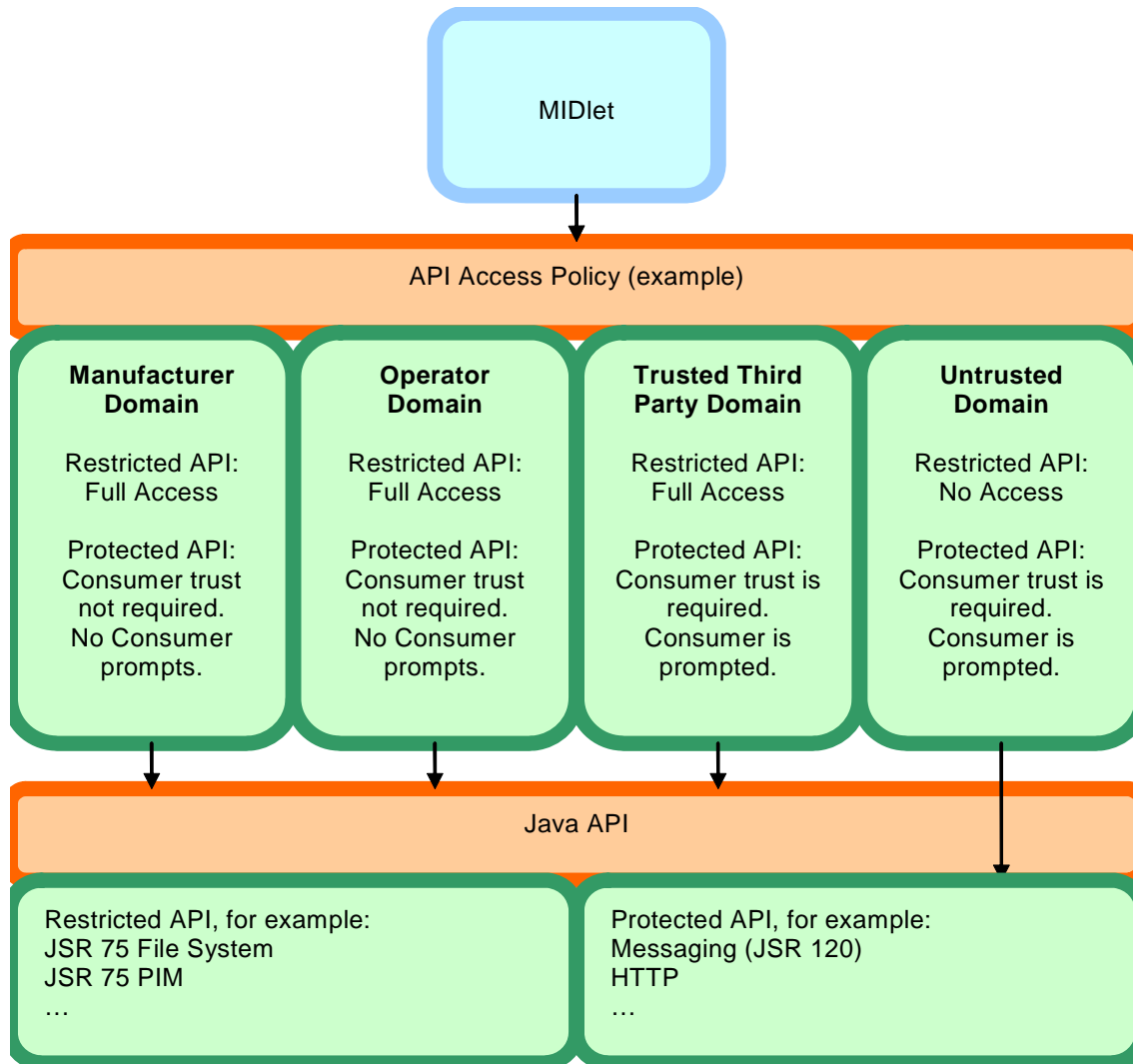


Figure 1: Example Domain and API Policy Model

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 by Motorola can be installed and run.

API access

Motorola handsets (retail/unbranded handsets) provide by default a trusted signed MIDlet with access to all sensitive APIs, as listed in Table 1. 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 the “Motorola API Device and Demo Applications Matrix,” in the Software Development Kit (SDK) associated with the handset.

Table 1: API Access Matrix

Restricted/Sensitive API/Function	Untrusted Access	Trusted Access
Network: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • javax.microedition.io.PushRegistry 	Yes (with consumer confirmation prompt)	Yes
Connectivity: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • javax.microedition.media.control.RecordControl • javax.microedition.media.control.VideoControl.getSnapshot 	Yes (with consumer confirmation prompt)	Yes

Table 1: API Access Matrix (Continued)

Restricted/Sensitive API/Function	Untrusted Access	Trusted Access
User Data Read: <ul style="list-style-type: none"> • javax.microedition.io.Connector.file.read • javax.microedition.pim.ContactList.read • javax.microedition.pim.EventList.read • javax.microedition.pim.ToDoList.read 	No	Yes
User Data Write: <ul style="list-style-type: none"> • javax.microedition.io.Connector.file.write • javax.microedition.pim.ContactList.write • javax.microedition.pim.EventList.write • javax.microedition.pim.ToDoList.write 	No	Yes
Location: <ul style="list-style-type: none"> • 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

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 is 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.

The result of this scenario is this consumer prompt:

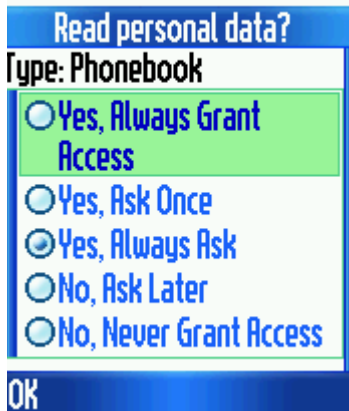


Figure 2: Consumer prompt for JSR-75 PIM API Access

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, depending on the response selected. 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 future instances. 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 you approach the specific operator for digital signing of your 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

MIDlet development is best done in stages:

- Concept and specification writing
- Initial development using a handset emulator for testing code
- Final testing on a real handset and project completion and acceptance
- Application delivery to a sales channel

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

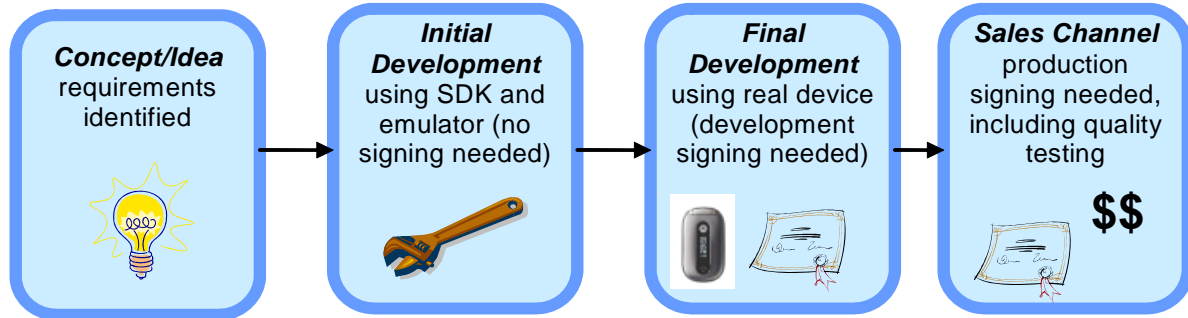


Figure 3: Digital Signing During MIDlet Development

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 not possible to test an untrusted MIDlet on a real handset 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 should test the MIDlet on a handset to ensure that it works 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 handset, the MIDlet must be trusted by way of a "limited" development signing certificate signature. For more information about Motorola development certificates, See "Development certificates" on page 24." The limitation usually takes the form of a restriction on the number of handsets onto which the MIDlet can be installed or an expiration time/date. The limitation prevents a potentially untested MIDlet from reaching full production status and being released.

Production signing

Whereas a developer certificate enables final testing on a handset 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 a trusted 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 handsets or subject to 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 handset.

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 into the JAD file manually (copy/paste) or via an Integrated Development Environment (IDE) tool. If you are using MOTODEV Studio for Java ME as your development environment, there is an Application Signing tool which automates digital signing of MIDlet suites for Motorola handsets. For more information on the Application Signing tool, See “Method #3 - Call Launchpad.exe from MOTODEV Java ME SDK” on page 41. 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 needs 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. A bound certificate is only used during development and testing of a MIDlet. This certificate restricts you 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 you can install the development-signed MIDlet. If you want to commercially deploy your application, you must obtain a production certificate. See “Production signing” on page 23.

Unique Identifier

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 handset-specific information and configurations. Two examples of this type of information are the International Mobile Equipment Identity (IMEI) and the Universal Identifier (UID). This data may be needed to perform specific tasks associated with the development of a new Java MIDlet.

One such example is the 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](#).

Motorola has a Config Tool, available via the MOTODEV Studio for Java ME SDK Launchpad, and also from MOTODEV Studio for Java ME. The tool is accessible several ways, depending on your development environment. The main purpose of the MOTODEV Config tool is to provide an easy way for you to read and write certain specific internal Motorola handset configurations such as the handset UID.



Figure 4: UID Extraction using the Motorola Config Tool

Outside of MOTODEV Studio, you can get the required information by using the UID Extraction Tool.

NOTE: The UID Extraction Tool will no longer be actively developed and the recommended way to get the UID is via the Motorola Config Tool.

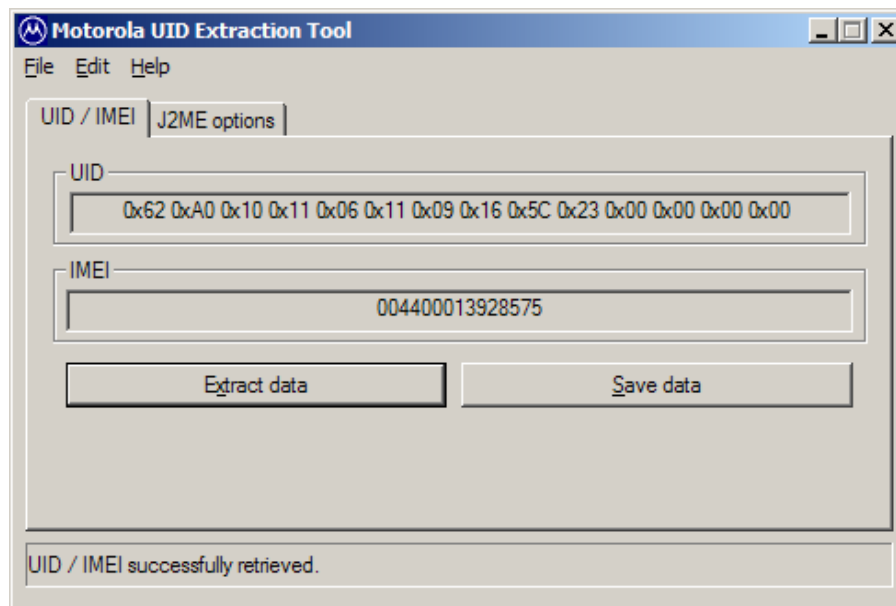


Figure 5: UID extraction using the Motorola UID Extraction Tool

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.

MIDlet production 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 controls access to restricted APIs. An application is assigned to a domain through the digital signature embedded into the MIDlet JAD file. During MIDlet installation, that digital signature is matched to a root certificate on the handset.

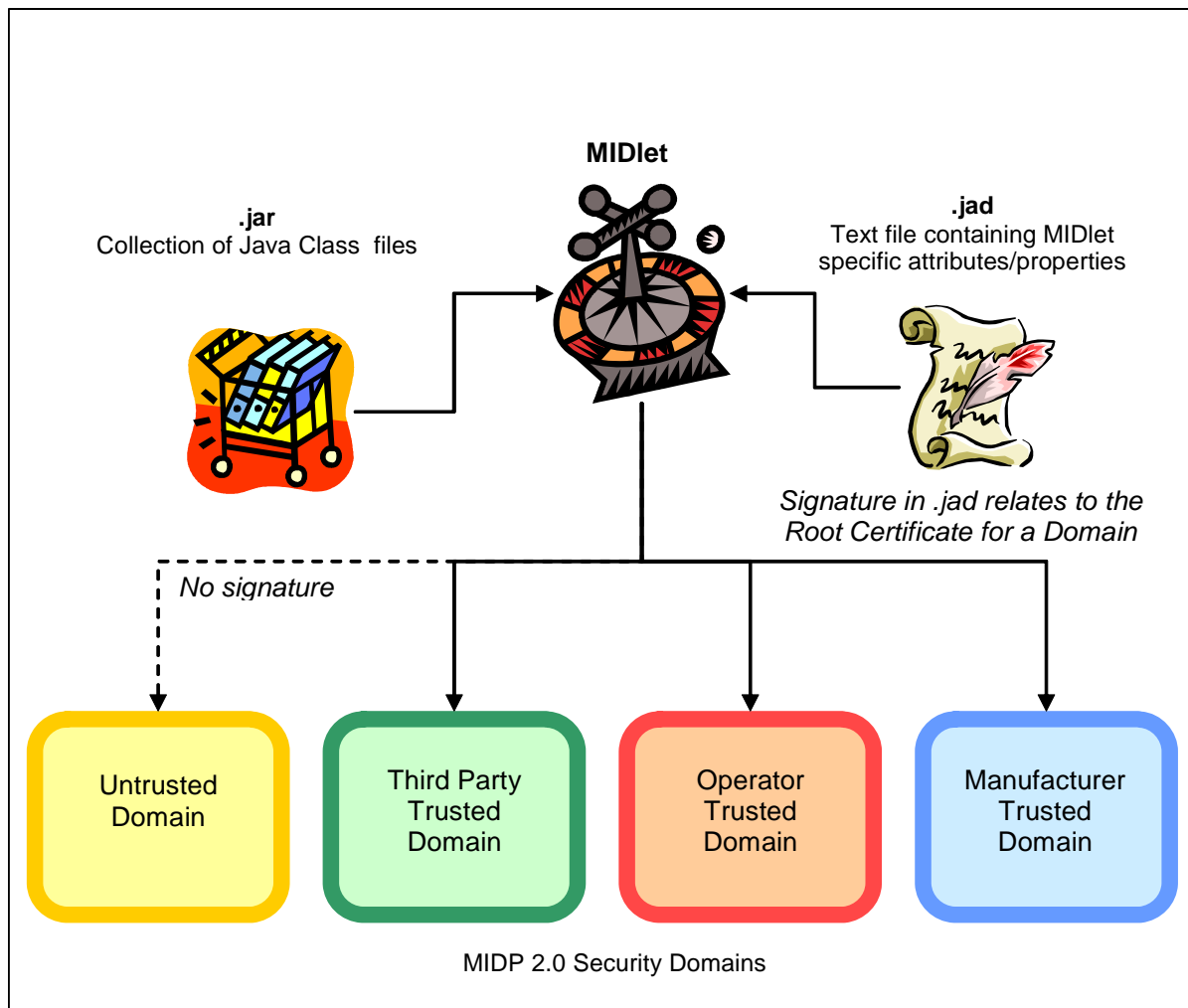


Figure 6: A signature relates a MIDlet to a MIDP 2.0 Security Domain

Choosing a signing authority

Not all MIDlets need to be signed. Depending on the required features as well as the desired consumer experience, you can choose among three signing authorities: the [Java Verified Program](#), operator's root certificate, a Motorola manufacturer's signature.

Java verified program

For the majority of MIDlets, MOTODEV recommends the [Java Verified Program](#). This is the trusted third party security domain. You 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 handsets on the [Java Verified Web site](#).

Operator root certificate

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

Motorola's trusted domain production code signing

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, and grants signatures only for a limited number of applications that demonstrate exceptional innovation and alignment with Motorola's goals of seamless mobility and iconic design. Because operators customize handsets, you must confirm with 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 limited source code review as part of Motorola's quality assurance process.

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

Table 2: 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 summary

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

- **Java Verified**—for signing into the Trusted Third Party domain
- **Operator**—for signing into the operator domain. Note that it is your responsibility to follow up with the target operator/carrier regarding this type of signing certificate.
- **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. Refer to the decision tree in the following figure to help you choose a signing authority.

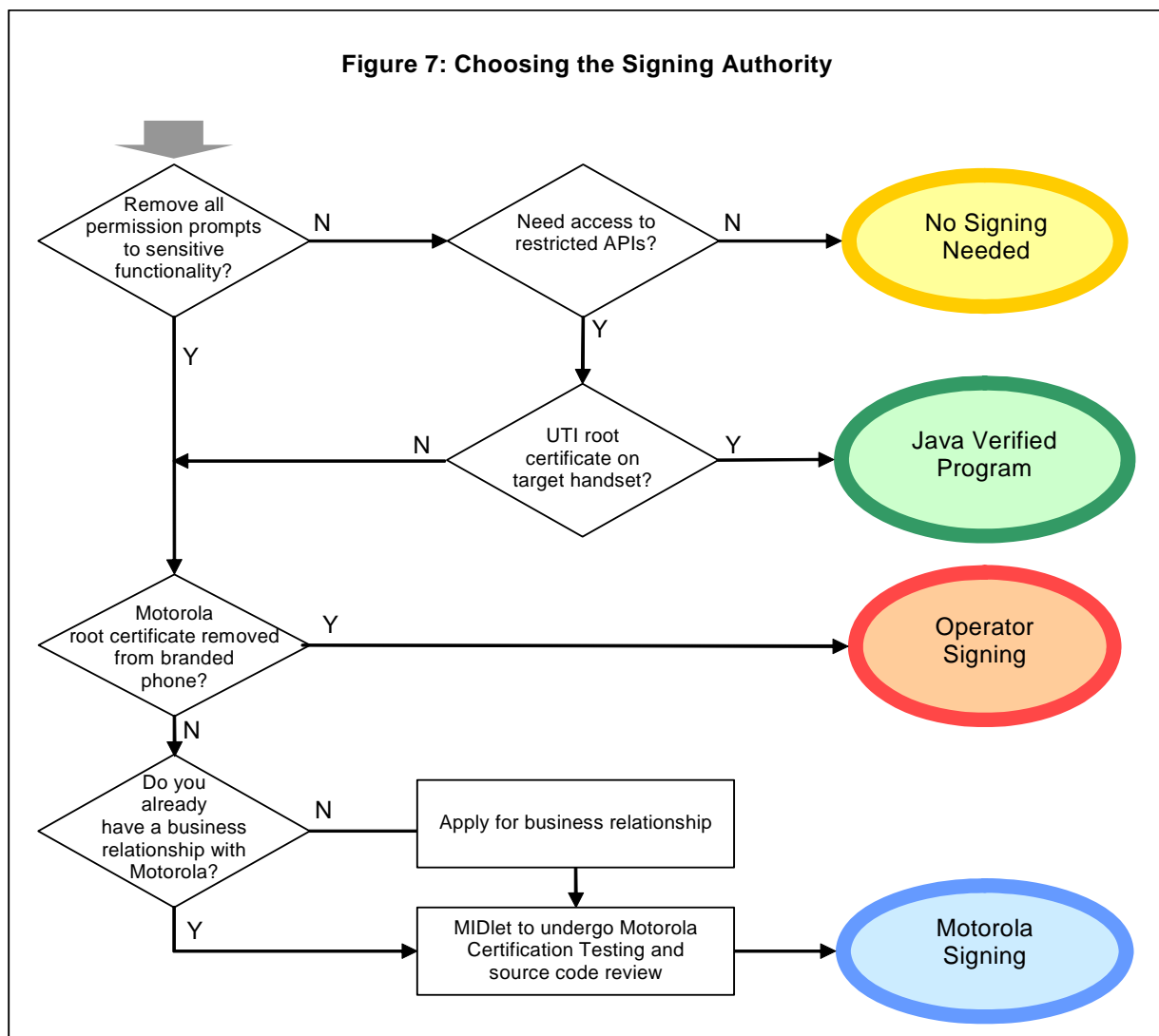


Figure 7: Choosing a Signing Authority

Motorola's 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), you can install and run MIDlets digitally signed by either the Java Verified Program or by Motorola.

By default, Motorola handsets (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 (SDK) associated with a specific handset.

Table 3: 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

Table 3: Trusted signed MIDlet access (default values are shown in parentheses)

API	Untrusted	Third-party Trusted	Operator	Manufacturer
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
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.EventList.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

Table 3: Trusted signed MIDlet access (default values are shown in parentheses)

API	Untrusted	Third-party Trusted	Operator	Manufacturer
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
SmartCardCommunications javax.microedition.apdu.sat <hr/> NOTE: javax.microedition.apdu.sat is not allowed for Trusted Third party nor for Manufacturer. <hr/> javax.microedition.apdu.aid	Not allowed	blanket (session) oneshot	Allow	Allow

Signing a MIDlet with a bound certificate

Motorola employs a development certificate on CLDC 1.1 products known as a “bound” certificate. A bound certificate is only used during development and testing of a MIDlet. This certificate restricts you 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 you can install the development-signed MIDlet. If you want to commercially deploy your application, you must obtain a production certificate. See “Production signing” on page 23.

This procedure allows signing of Java applications using bound certificates. A bound certificate has the serial number for the handset appended to the certificate format and the resulting MIDlet files are signed using the Motorola Mobile Devices Business Java Certificate Authority (CA). When the handset starts to

load this type of certificate, it identifies the bound tag, pulls the electronic number from the processor, and uses it to validate the signature of that certificate. Once this validation takes place, then the certificate is used to validate the signature of the JAR file and if it passes, installs the JAR file on the product.

NOTE: You may also copy the directory structure (all files) to another PC once the Bound Certificate is returned to you, which allows multiple developers to develop code and sign for a particular handset in that team. Thus the signing process is not tied to one development PC.

Prerequisites

- You, the developer, must register on the MOTODEV Web Portal for Technical Assistance.
- MIDlet development is an external project.
- You have a Motorola USB cable for communicating with the handset.
- You have a handset that has Java App Loader (JAL) enabled (optional).
- Ensure the handset model and UID(s) are retrieved from the handset using Motorola's recommended tools (e.g., the Config Tool or the UID Extraction Tool). See "UID extraction" on page 25.

Installation steps

MOTODEV Studio for Java ME is an integrated development environment (IDE) that includes an Application Signing tool which automates the digital signing of MIDlet suites for Motorola handsets. The Application Signing tool walks you through the step-by-step process. This tool can be found in the Help > MOTODEV Studio for Java ME Documentation > Studio for Java ME Tools > Application Signing. The Application Signing tool provides key generation, Certificate Signing Request (CSR) creation and frees you from dealing with the config.txt file, all with an easy to use Graphical User Interface (GUI).

If you are not using MOTODEV Studio for Java ME, you can do application signing using a manual process. The six step manual process is described next.

Step 1: Installing OpenSSL tools on your PC

Step 2: Creating a directory structure on your PC

Step 3: Creating the signing Key Pair

Step 4: Preparing the onfiguration file

Step 5: Creating a Certificate Signing Request and collecting UIDs

Step 6: Signing a MIDlet with a bound certificate

Step 1: Installing OpenSSL tools on your PC

1.1 Download and install OpenSSL package

Download the latest OpenSSL package from openssl.org: <http://www.openssl.org/related/binaries.html>. and install on the “C:” drive on your PC

1.2 Confirm directory structure

Confirm that a directory called “C:\openssl\” and that the following directories and sub-directories exist.

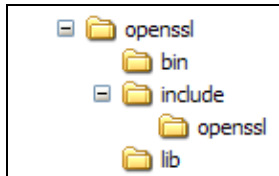


Figure 8: Directory Tree

1.3 Update the Path environment

Update the Path environment variable of your PC to include the following “C:\openssl\bin”.

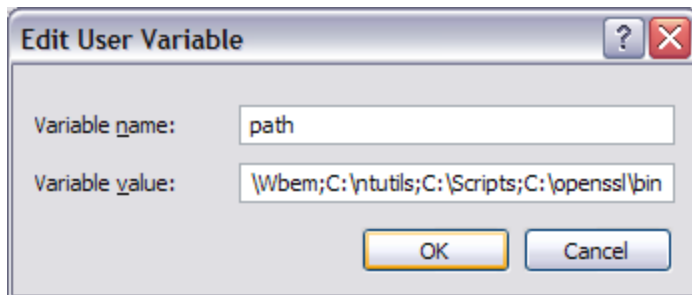


Figure 9: Edit User Variable

1.4 Verify program availability

Verify that the OpenSSL program is now available on your PC by typing “openssl version” in a command prompt window, as shown next.

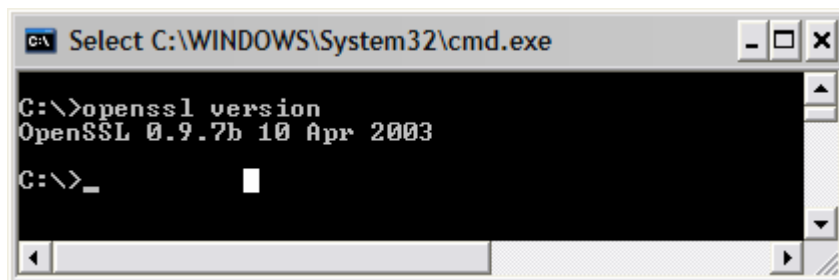


Figure 10: Step 2 Prompt

Step 2: Creating a directory structure on your PC

You are now required to create the following directory structure on your PC.



Figure 11: Step 2 Directory Tree

Directories are needed for the following:

- cert = for storage of certificate
- csr = for storage of certificate signing requests
- key = for storage of private keys
- sign= for storage of MIDlet signatures

Step 3: Generating the signing Key Pair

The following procedure describes how to generate a Key Pair, used to allow signing of the Development MIDlets. Change directory to “Bound_Certificate” and run the following command in a command window. Enter your company name in place of **<company>** and your own unique password in place of the **<password>** listed below.

```
openssl genrsa -out key\<company>.key -passout pass:<password> 1024
```

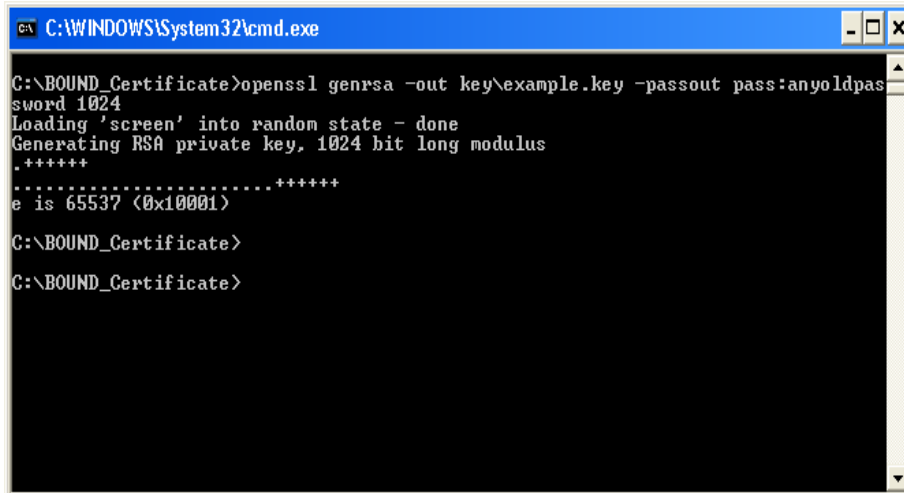


Figure 12: Generate a key

WARNING: It is also important to mention that the public key exponent must be set to 65537, which is the defacto Rural Service Area (RSA) exponent for most use cases in the world. If you use

the OpenSSL tool for making bound certificate requests, this value is used by default. However, if you use any other tool, you need to make sure the generated public key satisfies this requirement.

Step 4: Preparing the configuration file

4.1 Create a CSR

If you are using MOTODEV Studio for Java ME, follow the step by step instructions on how to create a CSR (Certificate Signing Request) as described in the online help from MOTODEV Studio for Java ME. Outside of MOTODEV Studio, follow the manual process shown to create a config.txt file and a CSR file.

4.2 Create a “config.txt” file

In the main directory “Bound_Certificate”, create a text file called “config.txt”.

Name ▲	Size	Type	Date Modified
cert		Folder	23/07/2004 12:16
csr		Folder	20/07/2004 15:28
key		Folder	28/01/2005 15:26
sign		Folder	28/01/2005 15:35
config.txt	1 KB	Text Document	08/07/2004 15:07

Figure 13: Directory Tree

4.3 Verify contents of config.txt file

Inside the “config.txt” file, ensure that the following is created (note that this file is case sensitive):

```
[ca]
default_ca = default
[default]
dir = .
new_certs_dir = $dir/cert
private_key = $dir/key
policy = policy_any
[policy_any]
countryName = supplied
stateOrProvinceName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
[req]
distinguished_name = req_dname
prompt = no
x509_extensions = v3_ca
[req_dname]
CN = Bound Certificate for <Name of the Developer Here>
C = <Country>
O = <Organization>
OU = <Organization Unit>
[v3_ca]
subjectKeyIdentifier = hash
basicConstraints = critical,CA:true
keyUsage = critical,digitalSignature,keyCertSign
```

Make sure that you replace the information contained within angle brackets (“<” and “>”) with real values. For example:

CN = Bound Certificate for <Name of the Developer>

C = <Country>

O = <Organization>

OU = <Organization Unit>

Becomes:

CN = Bound Certificate for John Doe

C = UK

O = XYZ Company

OU = ABC department

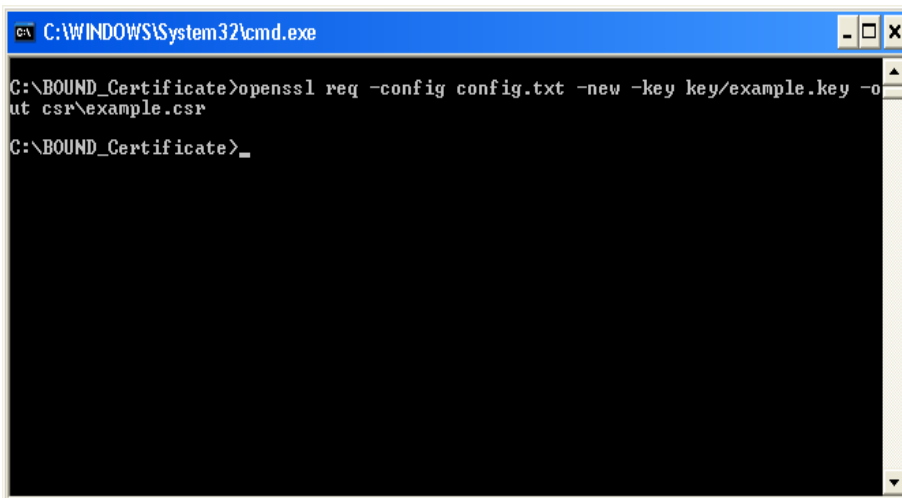
NOTE: Note that the Country value must conform to the ISO 3166 countries format. See the following site for details: <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html>

Step 5: Creating a Certificate Signing Request and collecting UIDs

5.1 Generate a CSR file

Perform the following command in the “Bound Certificate” directory, replacing <company> with your company name.

```
openssl req -config config.txt -new -key key/<company>.key -out csr/<company>.csr
```



```
C:\WINDOWS\System32\cmd.exe
C:\BOUND_Certificate>openssl req -config config.txt -new -key key/example.key -o
ut csr\example.csr
C:\BOUND_Certificate>_
```

Figure 14: Figure 7 Generate a CSR file

Possible problems seen at this stage in procedure include the following:

```
Error Message from OpenSSL command:

Problems making Certificate Request
2176:error:0D07A097:asn1 encoding routines:ASN1_mbstring_copy:string too
long:.\
crypto\asn1\a_mbstr.c:154:maxsize=2

Reason: The value of C in the config.txt is not ISO-3166 compliant. See Step
4: Prepare the Configuration File, for more information.
```

5.2 Extract the UID

Extract the UID from your development handset using the Config Tool. The Config Tool is available both from the MOTODEV Studio for Java ME or the standalone MOTODEV Java ME SDK. Both are available for you to download from the tools link at <http://developer.motorola.com>.

NOTE: In the past you may have used the UID Extraction tool to get the UID from your development handset. This may still work for older handsets. However, moving forward, newer handsets will use the Config Tool. Support for the UID Extraction tool will be phased out.

5.3 Run update manager

Ensure that you have the most up to date version of MOTODEV Studio or MOTODEV Java ME SDK by running the update manager from Help > Software Updates from MOTODEV Studio or Help > Update Manager from MOTODEV Java ME SDK.

5.4 Download USB drivers

Download the latest USB drivers from Motorola at <http://developer.motorola.com>.

5.5 Verify USB connection setting

Check that the development handset's USB connection setting is set to "Modem" and not "mass storage". Connect your development handset to your PC using a USB cable.

5.6 Run Config tool

Run Config Tool from the "Tools" link from either the MOTODEV Studio or the MOTODEV Java ME SDK. Click refresh to display the details of the currently connected handset. The Config Tool should be populated with information about the handset, including the UID data.

NOTE: If the drivers are not fully installed, you may need to wait at this point or remove and reconnect the handset, then click the refresh button again.

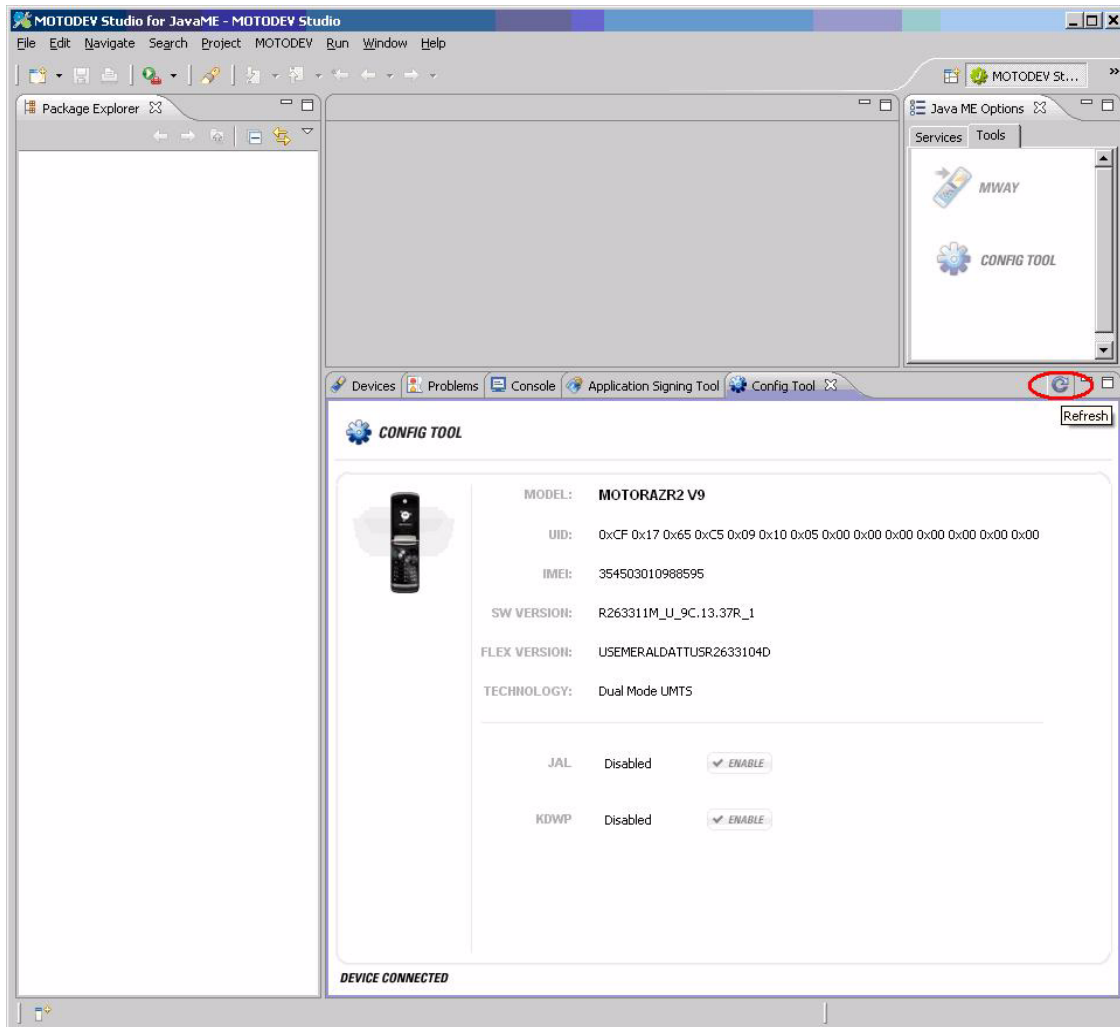


Figure 15: Config Tool UI from MOTODEV Studio

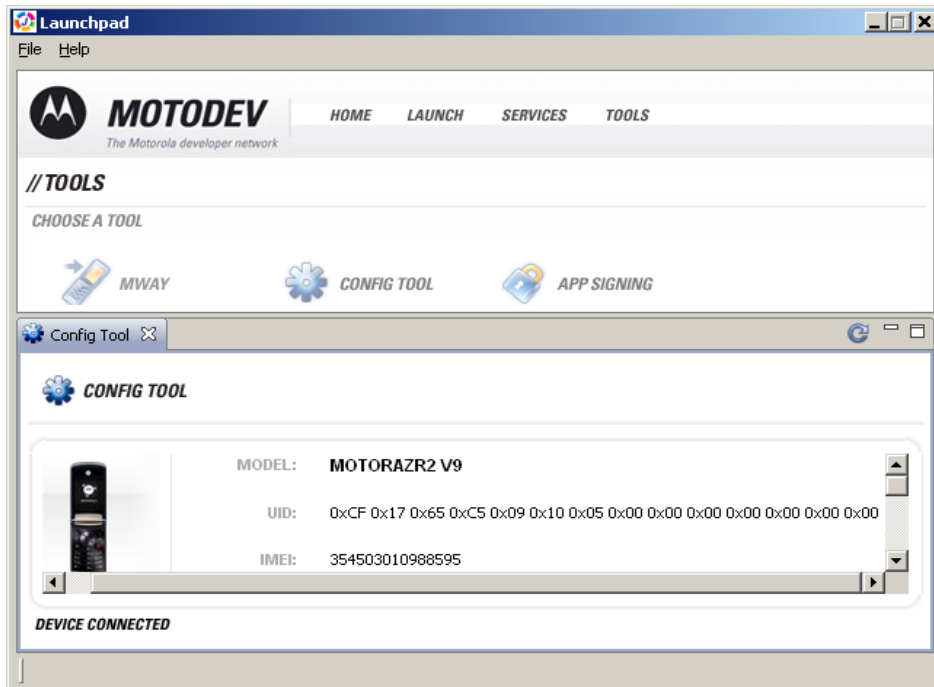


Figure 16: Config Tool view from the MOTODEV Java ME SDK Launchpad

5.7 Submit technical support incident

Submit the following information into a technical support incident on the MOTODEV web site:

- The “<company>.csr” file.
- A request text file containing the UIDs of the handsets you are using (maximum 15 UIDs). Follow the directions of FAQ 570 on the MOTODEV web site on how to create the Bound Certificate Request text file.

If the data provided is valid, you will receive a Bound Certificate within one to two days from Motorola.

Step 6: Signing a MIDlet with a bound certificate

Upon receiving a Bound Certificate file (*.crt) file from Motorola, you can sign your application in any one of four ways:

- Method #1 - use MOTODEV Studio for Java ME to sign your application
- Method #2 - call the underlying Java classes that are used by the MOTODEV Studio GUI from a command line
- Method #3 – call Launchpad.exe from MOTODEV Java ME SDK
- Method #4 - manually sign your application

Method #1 – Use MOTODEV Studio for Java ME

Using the MOTODEV Studio for Java ME GUI is the simplest and the most convenient way of signing your application.

Refer to the online help in MOTODEV Studio for Java ME for the most up-to-date step by step instructions on how to sign your MIDlet.

Method #2 – Call the underlying Java commands that are used by the MOTODEV Studio GUI from a command line

In certain situations, it might be useful to execute the underlying Java commands instead of using the GUI. This may be applicable if you frequently sign a large volume of applications and would rather include the commands in a script. You may also integrate these commands within your build environment if you already have an existing build system.

Again, refer to the online help in MOTODEV Studio for Java ME for the most up-to-date step by step instructions on how to sign your MIDlets from a command line interface.

Method #3 - Call Launchpad.exe from MOTODEV Java ME SDK

If you are just using the MOTODEV Java ME SDK and not the MOTODEV Studio for Java ME, the Application Signing tool is also accessible from that environment through the Launchpad application.

- From your MOTODEV Java ME SDK directory, double click launchpad.exe. This is a stand-alone GUI where you can easily access the tools and resources you need during application development.
- Click Tools to access the App Signing tool.

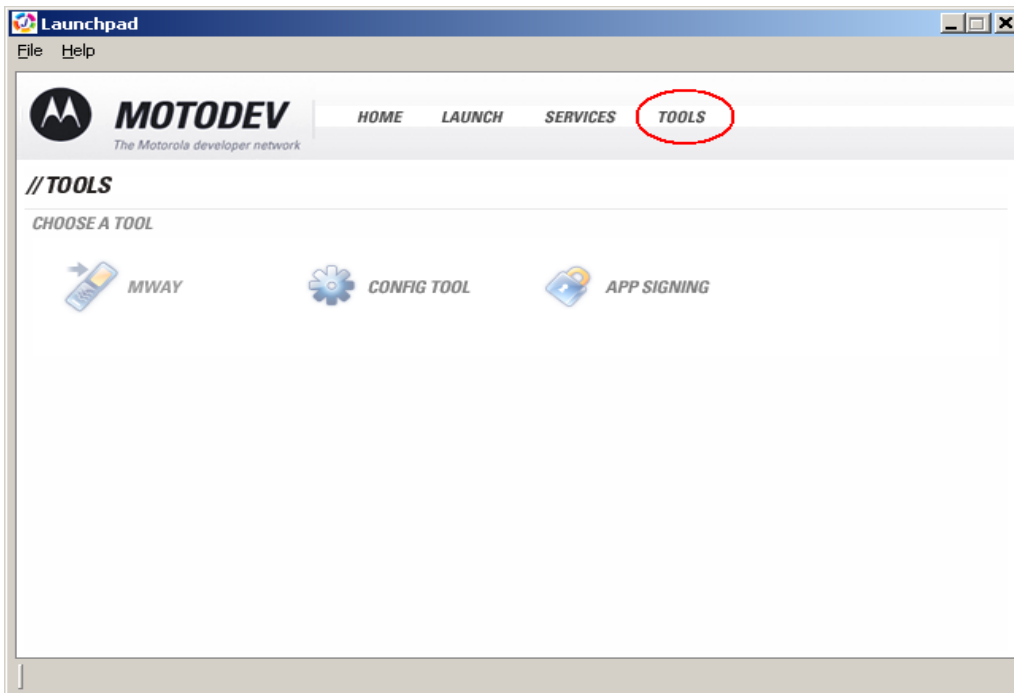


Figure 17: Launchpad from MOTODEV JAVA SDK

- This is the same tool as described in Method #1. Follow the detailed step by step instructions as described in the online help.

Method #4 - Manually sign your application

If you already have your own tools suite and are not using the MOTODEV Studio for Java ME, you may manually sign your application following the steps below. These steps can also be scripted and integrated into a build environment.

- 1 Upon receiving a Bound Certificate (.crt) file from Motorola, you should create a directory for the Motorola product(s) for which this certificate is used. Recall that you can submit up to 15 UIDs (that is, 15 handsets) for one Bound Certificate.

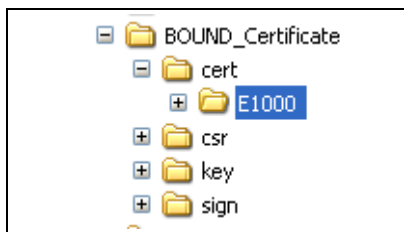


Figure 18: Directory Tree

- 2 Place the Bound Certificate inside this directory.
- 3 Open a command window in the Bound_Certificate Directory and run the following command to create a signature for the JAD file.

```
openssl dgst -sha1 -binary -out sign\sign.bin -sign key\openssl base64 -in sign\sign.bin -out sign\sign.b64
```

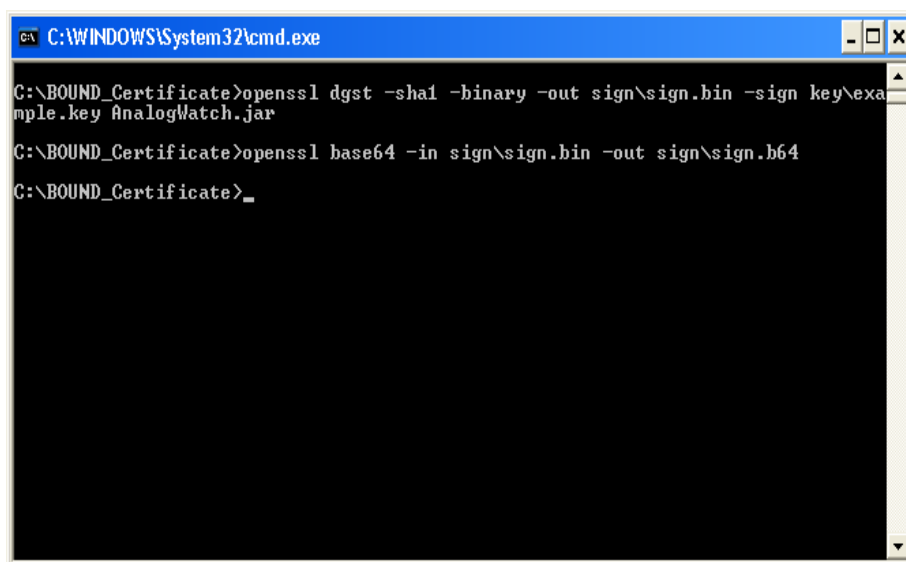


Figure 19: Prompt to create signature

- 4 Open the file sign.b64 and copy the content into the.JAD file for the MIDlet, making sure you add the attribute “**MIDlet-Jar-RSA-SHA1:**”, as shown here:

```
MIDlet-Jar-RSA-SHA1:bBJ6j/SrhhT0kUMhLpMzHH4p9uLLKF3I81SZdZw...
```

NOTE: The string must be in one line of text with all line breaks removed.

- 5 Copy the base64 encoded value of the bound certificate and insert this into the.JAD. To achieve this, change to the directory where the bound certificate is located (.cert).
- 6 Locate the (.cert) file and open it in Notepad., and copy the string between the BEGIN and END section in the following figure.

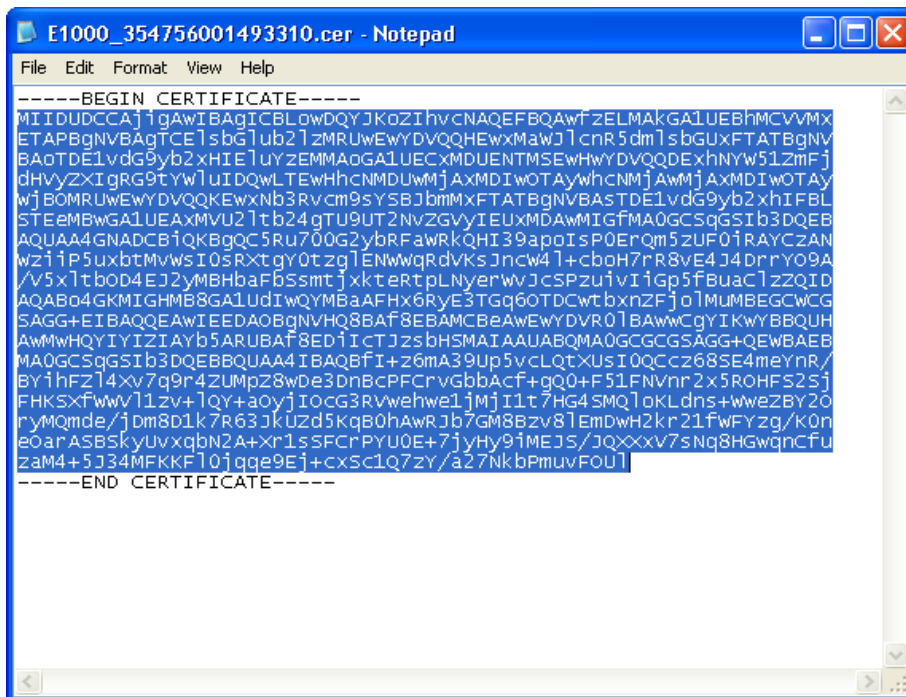


Figure 20: Certificate screen shot

- 7 Insert this into the.JAD file like so, with the attribute “**MIDlet-Certificate-1-1:**”.

```
MIDlet-Certificate-1-1: MIICoJCCAguGAWIBAgIBA...
```

NOTE: The string must be in one line of text with all line breaks removed.

You now have a signed MIDlet for the handset for which you provided a UID. You can now install the MIDlet on this handset.

Adding UIDs to an existing bound certificate

Bound Certificates can contain up to 15 UIDs and support up to 15 different handsets. To add extra handsets to the Bound Certificate, simply add the UID at the end of the list of UIDs in the text file and re-submit the text file, together with the .csr file.

If you decide to use a new text file with only the new UID(s), you will have to sign your MIDlet twice; once with the original Bound Certificate and once for the second Bound Certificate for the new handset(s). Combining as many UIDs as possible into one Bound Certificate will optimize your work flow and reduce the risk of signing the MIDlet with the incorrect certificate.

It is very important that you keep the UIDs of incompatible Motorola OS versions separate. Incompatibilities will require different Bound Certificate Request text files. See FAQ 570 on the MOTODEV web site for more information on Motorola OS versions and the associated Bound Certificate Request text files.

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, you are 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, you need 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 that you 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: *Motorola 3D API*

Recently, more phones are starting to support JSR-184 (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 capability and brings to the customer a new visual experience. It is expected that 3D games, or applications based on Mobile 3D Graphics, will become significantly more popular with the increasing number 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 1: 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 2: 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 3: 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
```

```

        -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 4: 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 5: Creating 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 21.

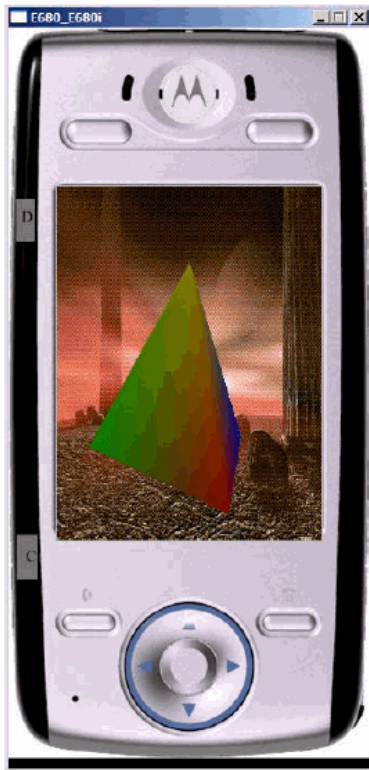


Figure 21: Tetrahedron object

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.

Chapter 5: Motorola 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.

iTAP is a text input method for mobile handsets that adapts to a user's pattern of communications to reliably predict and suggest entire sentences based on past usage. The iTAP system becomes more accurate over time and makes composing entire sentences and paragraphs, intuitive, easy, and personalized.

The iTAP system requires only one key touch to select a letter, and also proposes the next word you intend to add to your message or note. By using past text input and the context of where this text was entered, iTAP software goes far beyond simply remembering unique words, slang and abbreviations and suggests entire sentences.

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 3: 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 6: JSR-118 MIDP 2.0 Application Testing and Signing

The notion of trusted applications was introduced with Mobile Information Device Profile (MIDP) 2.0. Trusted applications are permitted to use APIs that are designated as sensitive and restricted. 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 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 digital signature from a signing authority. MIDP is designed to work on top of Connected Limited Device Configuration (CLDC) versions 1.0 and 1.1.

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 Device Feature Matrix, available on MOTODEV and also in MOTODEV Studio for Java ME and the MOTODEV Studio SDK.

Gaming API

The Gaming API provides a series of classes that enable rich gaming content for the handset. This API improves performance by minimizing the amount of work done in Java, decreasing application size. The Gaming API is structured to provide freedom in implementation, 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.

Platform Request API

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

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.

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

Multiple key press gaming support

Multi-button press support enhances the gaming experience by giving the user the ability to press two keys simultaneously so that the corresponding actions of both keys occur at the same time. An example would be moving to the right while firing at objects in a game.

The following sets of keys support multi-button key press 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

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

Table 4: 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 6: 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 22).



Figure 22: 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 7 shows the implementation of HTTPS

Code Sample 7: 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() );
        }
    }
}
```

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 8 is an example of a Push Registry implementation.

Code Sample 8: 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)) {

```



```

        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 10: 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()) {
                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 23).

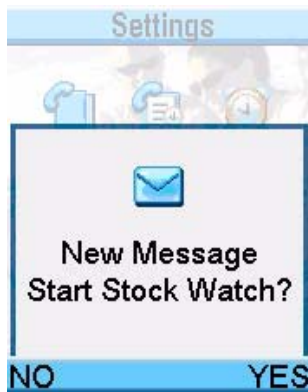


Figure 23: Run Intended Application Query

Table 5: 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.

Command class type priority, influence on placement

Our implementation of command class types assigns priority of command type over the application-defined priority for commands placement. This means that regardless of the priority set by the MIDlet, the BACK command has precedence over the SCREEN command. The application-defined priority is used to determine priority amongst several commands of the same type.

The priority between command types is the following (from higher to lower):

1. BACK
2. EXIT
3. CANCEL
4. STOP
5. OK
6. SCREEN
7. ITEM
8. HELP

Chapter 7: JSR-120 - 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) Short Message Service (SMS) on the target device.

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

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

- 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 6: 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 11 shows the implementation of the JSR-120 Wireless Messaging API

Code Sample 11: 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 12: 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 8: 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 13 is an example of the usage of the JSR-135 Mobile Media API:

Code Sample 13: 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 <code>getLevel</code> . 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`:

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

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`:

<code>getStopTime()</code>	Gets the last value successfully by <code>setStopTime</code> .
<code>setStopTime(long stopTime)</code>	Sets the media time at which you want the <code>Player</code> 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 7: Image Media

File Type	Codec	Functionality
JPEG	JPEG	Capture

Table 8: Image Media

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

Table 9: 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 10: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 11: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 12: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 13:Security Policy

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

Table 13: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 14: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.

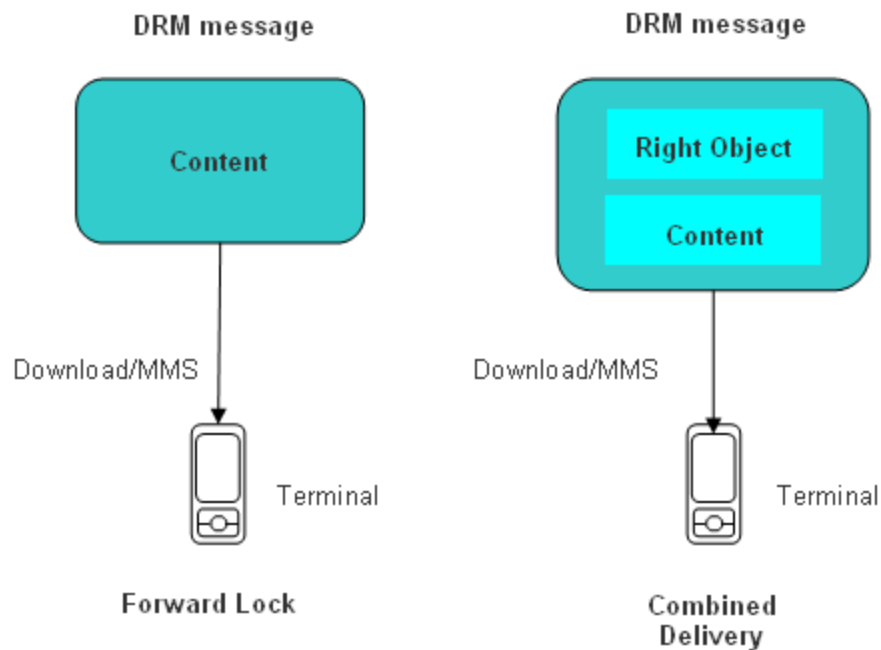


Figure 24: Forward Lock and Combined Delivery

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.

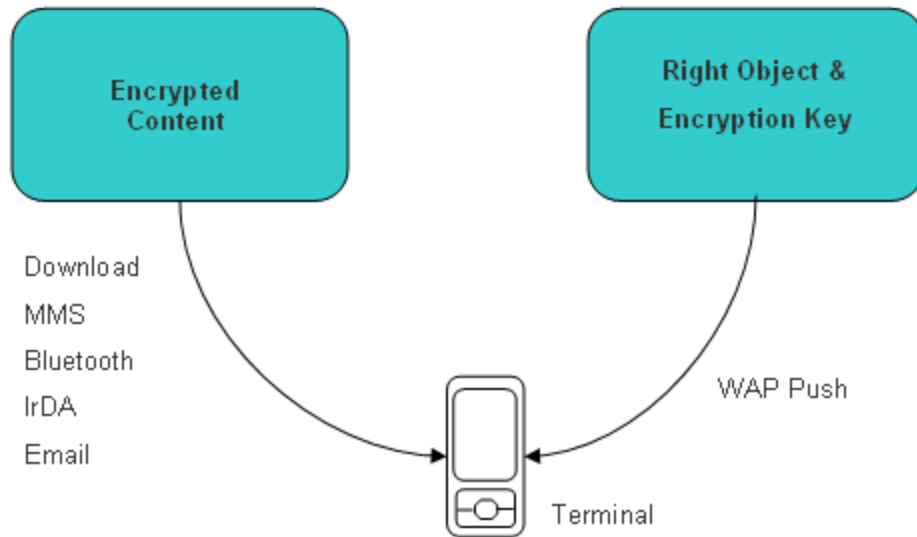


Figure 25: Separate Delivery

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.

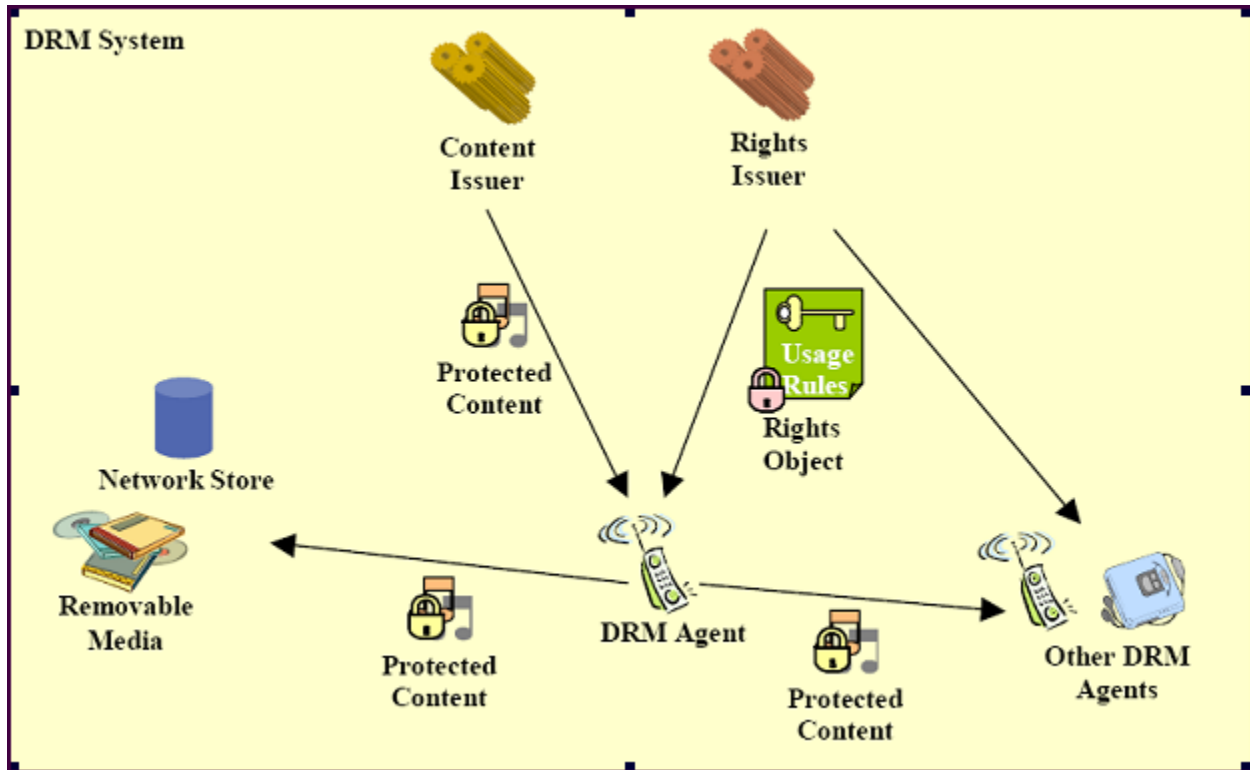


Figure 26: DRM 2.0 architecture diagram from OMA

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.

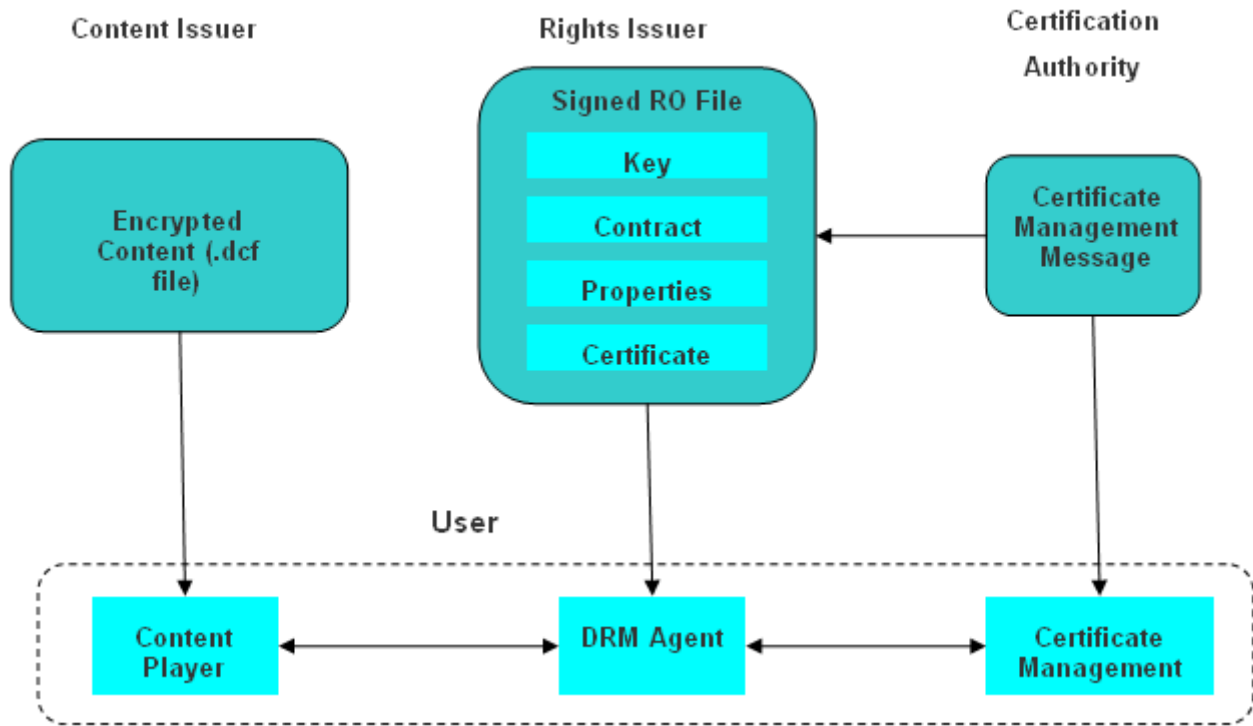


Figure 27: DRM 2.0 content download and use

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 15: 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 9: 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.

Appendix A: Key Mapping

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

Table 16: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 IcdUI.TextField or IcdUI.TextBox with PHONENUMBER constraint set)
Center Select	Select

Table 16: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 17 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 17: 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 <code>javax.microedition.midlet.MIDletclass</code> .	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 17: 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

Custom Motorola JAD attributes - *need to verify*

Many carriers and phone manufacturers have some specific custom JAD attributes. Motorola also provides some custom JAD attributes that are very useful for MIDlet applications.

Attribute Name	Attribute Description	JAR Manifest	JAD
Mot-MIDlet-Save-Location	Accepts the folder title in English. For example, you should specify what you see on the phone when English is selected. It also accepts four predefined values: Games, Tools, Multimedia, and WebAccess. These values correspond to internal MMA names regardless of the English title displayed.		
Mot-MIDlet-installTo	Accepts the internal main menu name as its value. The names are GamesFolder, Mutilmedia (with misspelling), Tools, and WebAccess. For user-defined folder, it will be “7”, “~8”, etc. No longer supported.		

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 18: 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 19 shows the error logs associated with downloading MIDlets.

Table 19: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 19: 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 20: 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:

<code>CellID</code>	The device's current Cell ID is returned during implementation.
<code>batterylevel</code>	The application's current battery level is returned, during implementation, as a percentage of full charge
<code>IMEI</code>	International Mobile Equipment Identity. The device's IMEI number is returned during implementation.
<code>MSISDN</code>	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](http://www.java-tips.org/java-me-tips/midp/how-to-retrieve-system-properties-in-a-midlet.html) 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 14: 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 21: 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.comports	(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.jtwn.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 22: 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 22: Motorola `getSystemProperty()` keys and their corresponding values (Continued)

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

Table 22: Motorola getSystemServiceProperty() 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	