MOTODEV

The Motorola developer network

Java ME Developer Guide for MOTOMAGX

1.0

Developer Guide

Java ME Developer Guide for MOTOMAGX

Version 1.0

March 2008

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

Motorola, Inc.

http://www.motorola.com

Contents

Chapter 1: Overview

Purpose and audience

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

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

Developer tools

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

MOTODEV Studio for Java ME

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

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

MOTODEV SDK for Java ME

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

Additional resources

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

Technical articles

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

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

Table 1: MOTODEV Technical Articles

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

Table 1: MOTODEV Technical Articles (Continued)

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

Table 1: MOTODEV Technical Articles (Continued)

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

Developer knowledge base

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

Other developer documentation

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

User guides

- Motorola SDK User Guide
- Motorola MIDway User Guide

API Device Matrix

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

Motorola-proprietary APIs

- Motorola Bluetooth Remote Control API
- Motorola Get URL
- Motorola PIM Enhancement API
- Motorola PIM Enhancement without ToDo, see Motorola PIM Enhancement API
- Motorola Scalable Image APIs
- Motorola Scalable JPG Image (deprecated, replaced by Motorola Scalable Image APIs)
- Motorola Scalable Image Enhancements, see Motorola Scalable Image APIs

---

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

Media guides

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

JSR specifications

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

- JSR 30 - CLDC 1.0 API
- JSR 75 - PIM API and fileConnection API
- JSR 82 - Java™ APIs for Bluetooth™ Wireless Technology
- JSR 118 - Mobile Information Device Profile (MIDP) 2.0 API
- JSR 120 - Wireless Messaging 1.1 API

- JSR 135 - Mobile Media API
- JSR 139 - CLDC 1.1 API
- JSR-172 - Web Services API
- JSR 177 - Security and Trust Services API
- JSR 179 - Location API for J2ME
- JSR 184 - Mobile 3D Graphics API
- JSR-185 - Java Technology for the Wireless Industry API
- JSR 205 - Wireless Messaging 2.0 API
- JSR-211 - Content Handler API
- JSR-226 - Scalable 2D Vector Graphics API

Supported handsets

Included here is a list of the supported MOTOMAGX 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 MOTOMAGX Handsets

	Supported MOTOMAGX Handsets
A	A728, A760, A768, A780, A910, A1200 (MOTOMING A1200)
E	E680
M	MOTOMING A1200, MOTORAZR^2 V8, MOTOROKR E2, MOTOROKR E6, MOTOROKR U9, MOTOROKR Z6
Z	Z6 (MOTOROKR Z6)

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 145.

The following messages can appear during download:

- If the JAR file size does not match the size specified in the JAD, the handset displays "The installation you are attempting failed due to an invalid file type. Please try again." Upon pressing “OK”, the handset goes back to the browser.
- If the MANIFEST file is wrong, the handset displays "The installation you are attempting failed due to an invalid file type. Please try again." Upon pressing “OK”, the handset goes back to the browser.
- If the JAD does not contain the mandatory attributes, the handset displays "The installation you are attempting failed due to an invalid file type. Please try again." The handset goes back to the browser upon pressing “OK”.

- 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. For information about "Memory Full" and other error codes, see "Status and Error Codes" on page 145.

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.
- 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.
- 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.

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: Background Execution of MIDlets

The Mobile Device Information Profile (MIDP) version 2.0 for Java Micro Edition (Java™ ME) specification defines the architecture and associated Application Program Interfaces (APIs) required to enable development of Java ME applications (MIDlets) that can be run on Motorola Mobile Devices. The MIDP profile was developed with the flexibility to allow device manufacturers and carriers the ability to customize their application environment based on the needs and capabilities of their devices.

In a continuing effort to improve the experience on Motorola mobile devices, the MOTOMAGX (Linux OS) platform utilizes the flexibility allowed by the MIDP specification to provide new Java capabilities that are not available on devices running the Motorola proprietary operating system. One of the key functional differences is the support for background execution of MIDlets. This new capability does, however, require MIDlet developers to understand the implementation of the background feature and take the steps necessary to ensure the MIDlet produces the desired experience.

Motorola has two major mobile platforms, one uses the Motorola Proprietary Operating System, called Motorola OS; the other uses an embedded Linux Operating System, called MOTOMAGX. Typical Motorola OS products include the MOTORAZR V3, MOTORAZR V3x, MOTOKRZR K1, MOTOSLVR L7, and typical MOTOMAGX products include the A780, E680, MOTOMING A1200, MOTOROKR E2, MOTOROKR Z6, MOTORAZR^2 V8, and MOTOROKR U9.

This chapter outlines the changes in the KJava Virtual Machine (KVM) architecture which allow a MIDlet to run in the background on the MOTOMAGX platform and provides information to help developers create MIDlets for MOTOMAGX.

MIDlet lifecycle in MIDP 2.0 specification

JSR-118, Mobile Device Information Profile (MIDP) version 2.0 for Java ME specification, defines the architecture and associated APIs required, enabling development of MIDlets that can be run on Motorola mobile devices. Both Motorola OS and MOTOMAGX handsets follow the specification, but the implementation detail is different. The text below describes the Active-Paused-Destroyed states and how each state is entered in Motorola OS, and MOTOMAGX devices.

According to the specification, a MIDlet can only have four states:

- Not running
- Active
- Paused
- Destroyed

A running MIDlet can have the following three states:

- Active

- Paused
- Destroyed

Active state

The MIDlet is functioning normally; it is running in the foreground and has focus. This state is entered when:

- The MIDlet application is started from the device user interface (UI), the MIDlet.startApp() method will be called.
- The Application Management Software (AMS) menu calls the MIDlet.startApp() method.

NOTE: In MOTOMAGX devices (not Motorola OS devices), the `startApp()` method can be called when the MIDlet is active (for example when the MIDlet is moved form background to foreground execution mode.

Paused state

The MIDlet is initialized and is inactive or at rest. In this state, the MIDlet is not running in the background because it is suspended. It should not be holding or using any shared resources. In Motorola OS phones, the system completely suspends the KVM when the MIDlet is in a paused state. In MOTOMAGX phones, the MIDlet must include instructions to stop threads and release resources, otherwise the MIDlet will continue to execute in the background. This state is entered under the following scenarios:

- From the Active state after the MIDlet.pauseApp() method is called from the AMS and returns successfully or, the MIDlet.notifyPaused() method returns successfully to the MIDlet.
- In Motorola OS phones, paused from a running state by the user pressing the red END key followed by selecting the “Suspend” option from the AMS menu.
- In MOTOMAGX phones, touch screen phones such as the A780, E680, A1200 do not have the AMS menu. Only keypad devices such as the ROKR Z6 and ROKR E2 have the AMS menu in it. The pauseApp() method is called anytime the MIDlet loses focus.
- Paused from a running state by the user closing the handset flip (clamshell style devices only) or by pressing the red END key. The MIDlet may not be completely paused – it is the MIDlet developers responsibility to create MIDlet suspending functionality if background execution is not required in the MIDlet.
- Paused from a running state by an interruption event such as an incoming phone call.

Destroyed state

The MIDlet has released all of its resources and terminated. This state is entered under the following scenarios:

- From the Active or Paused state, after either the MIDlet.detroyApp() method or the MIDlet.notifyDestroyed() method is called and returns successfully.
- In Motorola OS phones, a MIDlet is destroyed from a running state by pressing the red END key followed by selecting ‘End’ from the AMS menu. The touch screen MOTOMAGX phones do not have the AMS menu. On these touch screen phones, when the red END key is pressed, the MIDlet will lose focus and will not be suspended.
- Destroyed from a running state via a user directive from the MIDlet menu.
- Destroyed from a running state via abnormal MIDlet termination.
- Destroyed form a suspended state due to the device being powered off.

The MIDlet lifecycle state machine is shown next.

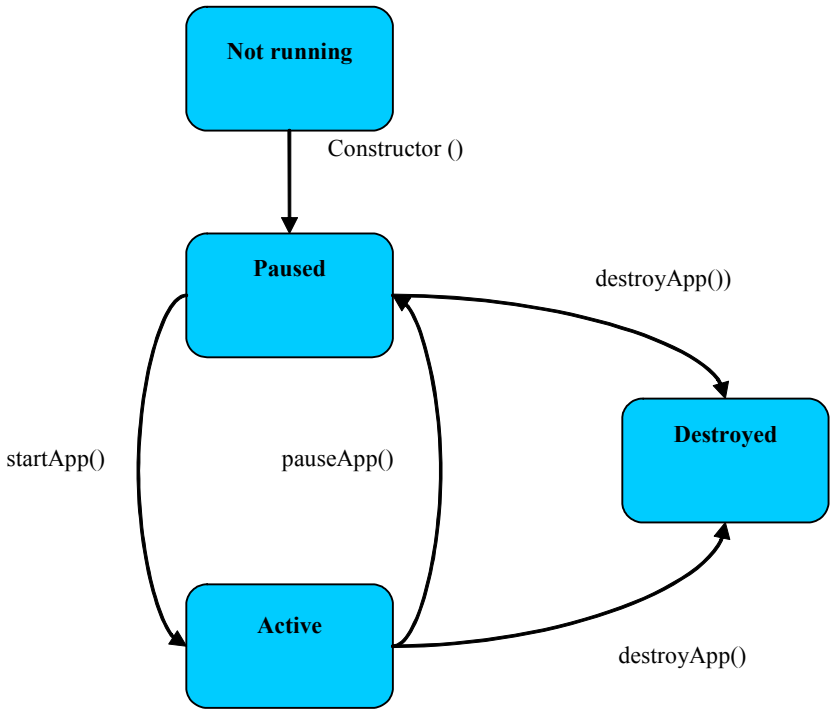


Figure 1: MIDlet lifecycle state machine

AMS (Application Management Software) is a part of the device operating system software that manages applications. It maintains the MIDlet state and directs the MIDlet through state changes. The `startApp()`, `pauseApp()` and `destroyApp()` methods are only called if the AMS finds the state change is necessary for that MIDlet. For example, in the Paused state, the `notifyPaused()` function does not trigger the `pauseApp()` function.

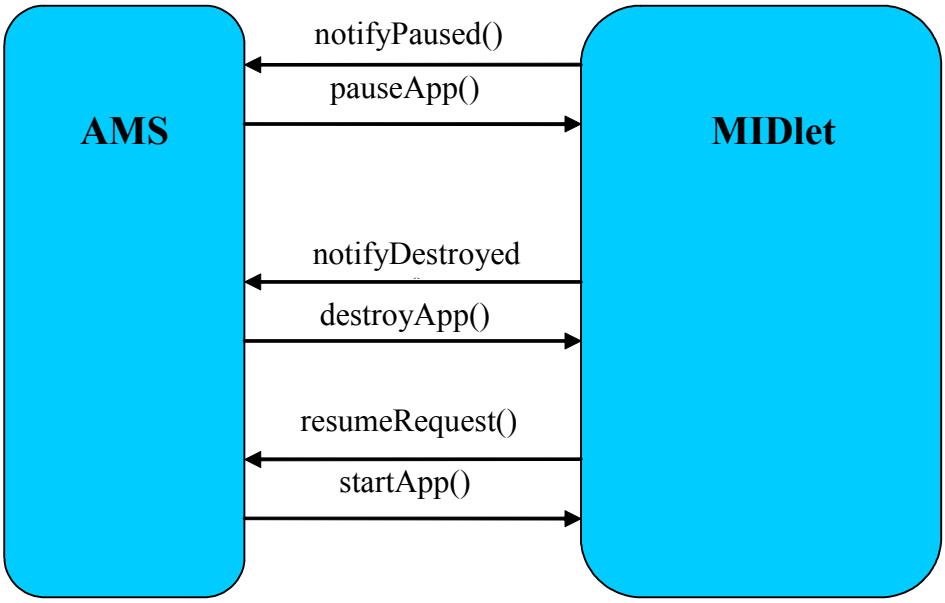


Figure 2: Interactivity between AMS and MIDlet

MIDlet background execution in MOTOMAGX phones

Following the paradigm of Linux-Java to provide a multi-tasking environment, devices developed on the MOTOMAGX platform allow MIDlets to continue running in the background while the device is being used for other activities. This permits an application such as an email client, music player, or an instant messaging application to continue running in the background, keep necessary network resources, and even play sounds or music while not in focus in the foreground.

Entering background mode

By definition, once the `pauseApp()` method is called, the MIDlet is in the Paused state. Per the MIDP 2.0 specification, a MIDlet "should not be holding or using any shared resources". It is the responsibility of the MIDlet to release unwanted resources when it is "paused". However, since this is defined in the specification as a recommended practice and not a mandatory requirement, a MIDlet may retain resources necessary to allow it to function. MIDlets that function in this way are in "background execution mode".

In the MOTOMAGX devices, MIDlets are able to run in the background. A MIDlet is informed that it is being placed in the background via the `pauseApp()` method. If the MIDlet does not pause itself or release any resources, the MIDlet will continue running in the background. If resources such as audio are being utilized by the MIDlet and are needed by a higher priority system function (such as a voice call), the resources will be taken away from the MIDlet to service the system requirement. In the MOTOMAGX phones, the KVM

process has lower priority than the native application processes such as voice calls.

Paused state behavior OS differences

In Motorola OS phones, when a MIDlet is suspended the `pauseApp()` method is called first, the MIDlet UI is hidden and the KVM process is suspended. Any thread and the `pauseApp()` method in the main MIDlet thread must finish all work within five seconds, otherwise, after five seconds the AMS will kill the active thread or the whole MIDlet thread respectively. Thus a MIDlet cannot be set to run in the background without focus and application multi-tasking is not allowed.

In MOTOMAGX devices, when a MIDlet loses focus (lost focus means that the MIDlet UI is not the top screen), the `pauseApp()` method is called. However, the KVM process continues to run in the background without focus and thus allows application multi-tasking. The MIDlet is able to continue running in the background while the device is being used for other activities. This permits an application such as an email client, a music player, or an instant messaging application to continue running in the background, retaining necessary network resources, and even play sounds or music while it is not in the foreground of the device.

In the Motorola OS phones, since the KVM process is completely suspended in the paused state, the `resumeRequest()` method cannot trigger the `startApp()` method in background.

In MOTOMAGX phones, the `resumeRequest()` method will trigger the `startApp()` method and transfer the MIDlet from a paused state to an active state, but it cannot bring the MIDlet to the foreground because it cannot get the display resource from a higher priority process.

The following is a "counter MIDlet" example source code to illustrate the behavior difference between the Motorola OS and MOTOMAGX devices.

```
protected void pauseApp() {
        form.deleteAll();
        counter = 0;
        form.append("pauseApp!\n");
        for(int i=0;i<10;i++){
                try{
                        Thread.sleep(1000);
                        counter++;
                        form.append("counter: "+counter+"\n");
                }
                catch(Exception e){
                        System.out.println(e.toString());
                }
        }
}
```

Code Sample 1: Counter MIDlet

In Motorola OS devices, when the user presses the red END key, the `pauseApp()method` is called first and the counter value is printed on the form, but after five seconds, the thread is killed by the AMS.

In the MOTOMAGX devices, when the user presses the “Home” button (on the A780 device for example), the MIDlet loses focus and the MIDlet UI is hidden. The `pauseApp()` method is then called and the MIDlet remains running in the background. When the MIDlet regains focus, all the counter values are displayed on the form.

The functionality that allows a MIDlet to run in the background can be simulated via the emulator in the Motorola SDK for MOTOMAGX products. The SDK emulator can simulate a network event, such as an incoming call, which generates an interruption and pushes the MIDlet to a paused state.

To use this feature:

1 Click on the emulator title bar

2 Click on the Network Events submenu

3 Choose the Pause option

To exit the pause mode, click again in the Network Events option and choose Resume.

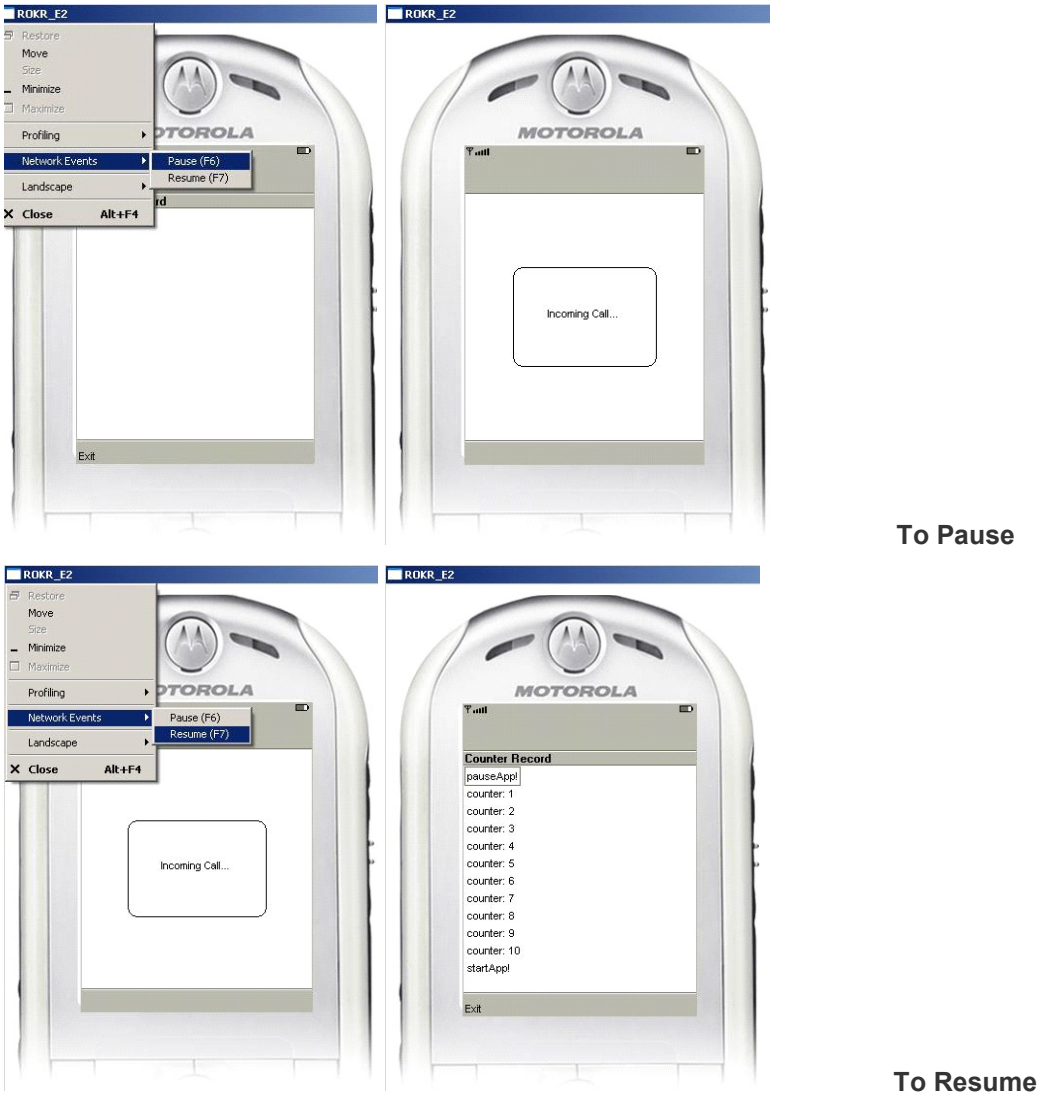


Figure 3: Simulate MIDlet background running in SDK emulator

Flip behaviors

On Motorola OS devices it is possible to instruct the AMS to ignore a flip close/open event and allow a MIDlet to continue to run unaffected by the flip operation. To do this in a Motorola OS device the JAD attribute “FlipInsensitive” can be used and set to “True”. The following line can be added to the JAD file:

```
FlipInsensitive: True
```

NOTE: On the MOTOMAGX devices this behavior is currently not available. It is planned and will be available in the near future. Please check the MOTODEV website at http://developer.motorola.com for news on when this feature is available in the Motorola Linux OS devices.

This means that on MOTOMAGX devices when the flip is closed, the MIDlet loses focus (it can still be running in the background) and the user has to bring the focus back to the MIDlet when the flip is reopened. There is visual indication to the user that the MIDlet is running in the background by way of a coffee cup icon in the status bar, at the top of the device screen.

Slider behaviors

On Motorola OS devices, there is a slider behaviour setting in the “Initial Settings” menu called “Slide Closed” which allows the user to control the effect the slider has on running applications. The MIDlet will either continue to run normally (if the setting is “Continue Task”) or be suspended with a call to the `pauseApp()` method (if the setting is “End Task”) when the slider is closed during MIDlet execution depending on this device setting.

On the MOTOMAGX devices, a slider close operation forces the MIDlet into the background and the pauseApp() method is called. The MIDlet can continue execution without focus. The MIDlet regains focus when the user selects it from the Java Application menu. Its startApp() method will then be called when it gains focus.

MIDlet development recommendations

The MIDlet developer should keep this behavior in mind and code a MIDlet to accommodate all possible scenarios. One such example is an incoming voice call which results in resources for a multi media player being taken from a running MIDlet and given to the system. Failing to take this into account correctly will result in a poor user experience or incorrect MIDlet behavior. In this example, the MIDlet may not continue to play audio when the multi media resources are returned to the MIDlet from the system.

Since the MIDP specification was created to support a minimum set of functionality that may or may not support background execution of MIDlets, special care must be taken when implementing a MIDlet designed to execute in the background. To ensure a good, reliable, user-friendly experience, the recommendations listed in the next section must be followed closely when developing MIDlets for a MOTOMAGX devices.

Pausing a MIDlet if background mode is not needed

For developers of games or other applications that do not need background mode execution, it is strongly recommended that the guidelines defined in the MIDP 2 specification be followed. This implies that when the `pauseApp()` method is called by the system, the MIDlet must pause the game or application, release resources (for example, audio) that are not needed while the game is paused, and take any necessary steps to preserve system or user data. Unlike the Java environment provided by devices running

Motorola's proprietary operating system, the MOTOMAGX platform will not automatically pause the execution of Java MIDlets based on calling the `pauseApp()` method. It is the responsibility of the MIDlet developer to stop the execution of the application once the device signals the MIDlet to pause.

Consequently, on resuming the MIDlet it is necessary to re-instate these resources and restore system or user data to the exact same state, at the point of suspension, in order to provide a good user experience. This is done by utilizing the `startApp()` MIDlet lifecycle method. The MIDlet can then continue its execution cycle.

Utilizing background mode

For developers wishing to make use of background mode on the MOTOMAGX platform, the MIDlet must be designed with the understanding that a `pauseApp()` request is passed to the MIDlet when the MIDlet is placed in the background. This call can be made as a result of a number of reasons such as incoming voice calls or flip/slider closure. The user can also manually put the application in the background by pressing the red END key and selecting the option from the native pop-up menu to return to IDLE with the MIDlet running. The `pauseApp()` method will be called once the background execution (IDLE) option is selected.

A MIDlet designed to run in the background must respond to the `pauseApp()` method call as required in the MIDP 2.0 specification, but can continue to utilize the resources it needs to perform necessary functions in the background. Since the MIDlet is in the Paused state once it responds to the `pauseApp()` method, it is highly recommended that any unnecessary processing be stopped, such as displaying updates. This will also help provide a better overall user experience on the mobile device.

The MIDlet's `startApp()` method is called to bring the MIDlet back to the Active state. When this method call is received, the MIDlet should request any necessary resources, resume updates to the display, and continue executing in the foreground of the device.

Audio resource handling

Executing in background mode does not guarantee audio resources will always be available. If a higher priority task occurs on the phone, such as an incoming phone call , the audio resources may be taken away (in order to play a ring tone for example). The system informs the MIDlet that the audio is unavailable via a `PlayerListener.DEVICE_UNAVAILABLE` event. The MIDlet is informed when audio resources are made available again via a `PlayerListener.DEVICE_AVAILABLE` event. The MIDlet should be able to process the event and take necessary actions to provide an optimal user experience. This could include pausing and restarting audio, stopping streaming audio, or replaying notifications once the audio resources are available again.

If the MIDlet is not concerned with the loss of audio resources while in background mode, it is highly recommended that the MIDlet again request audio resources once the MIDlet is placed back into the Active state (meaning the `startApp()` method is called). This should be done even if the audio resources were available at the time when the `pauseApp()` method was called.

If audio resources are not available when the MIDlet starts, it is the MIDlet's responsibility to provide any error messages to the user and/or terminate itself.

Network resource handling

Executing in background mode does not guarantee network resources will always be available. If network resources are lost for any reason, while the MIDlet is executing in the background, an exception will be thrown when the MIDlet attempts to read or write over the lost connection.

Threads

Many MIDlet developers create multi-threaded MIDlets. These threads can be used to control a network connection, a media player, a canvas, etc. When entering the paused state, any threads additional to the main MIDlet thread should be stopped (if not required in a MIDlet that is background executing). When leaving the paused state and returning to a normal running state, any stopped threads should be restarted again. This can be done by toggling a background running flag in the lifecycle methods `pauseApp()`/ `startApp()` and then test for this flag in the thread loop, as shown in the following ecode sample.

```
Thread_Run() {
    While(true) {
        If (is_in_background) {
            Wait(); // waiting for notification to wake up
            …
        }

        else {
            // running thread code
        }
    }
}
```

Alternatively, the same background running flag can be used so the thread can be slowed and unload the CPU as shown in the following example.

```
Thread_Run() {
    While(true) {
        If (is_in_background) {
            Sleep(1000); // manually slow the thread and unload the CPU
        }
        // running thread code
                }
)
```

Timer tasks

The `TimerTask` is often used by game developers and because it implements the Runnable Interface, it is recommended to suspend/cancel the object when the lifecycle method `pauseApp()` is called. The `TimerTask` object can be restarted in the lifecycle method `startApp()` as follows.

```
pauseApp() {
    ...
    working_timer.cancel(); //cancel task, renew when startApp() is called
    ...
}
```

Alternatively, the TimerTask can be rescheduled for some later time thus relieving the CPU of work.

```
pauseApp() {
    ...
    working_timer.schedule( timertask, delay, LONGER_PERIOD);
    // re-schedule the timer with long period to unload the CPU
    …
}
```

Record Management Store

MIDlets often store “persistent” data in the Record Management Store (RMS). This data could for example be game information that the MIDlet must not lose between invocations such as a high score table or game state. In the event of an abnormal MIDlet termination when in background execution mode, it is recommended that the RMS is closed and resources are released when a MIDlet enters a paused state. These resources should then be reallocated when they are needed by the MIDlet or when the MIDlet leaves the paused state and returns to a normal running state.

Best practice example for background mode

A Music Player example is provided to illustrate how to use the background execution feature in MOTOMAGX devices. In this example, the MIDlet can select to play the music in the background or not. The UI thread and music player thread are independent of the MIDlet main thread and the `DEVICE_AVAILABLE and DEVICE_UNAVAILABLE` events are handled correctly. The job of `startApp()` and `pauseApp()` is to control the threads and reset flags which indicate MIDlet states.

Figure 4: Music Player example for utilizing background mode

Potential conflicts

Many MIDlet developers use methods that could cause potential conflicts.

- The showNotify() and hideNotify() methods to control pause/suspend behavior of MIDlets utilizing a Canvas or GameCanvas.
- The Displayable.isShown() method to determine if a Form or TextBox object is visible to the user.

These methods can be used by the developer to control the MIDlet with respect to lifecycle states but care must be taken to ensure that there are no resource conflicts introduced between the `pauseApp()` and `hideNotify()` methods and the `startApp()` and `showNotify()` methods.

- Often pauseApp() is used to deallocate resources and objects as is hideNotify().
- Subsequently startApp() is used to reallocate resources and objects as is showNotify()

Summary

In MOTOMAGX devices, Motorola has implemented a lifecycle architecture that differs slightly from the Motorola OS devices. The goal of this flexible design is to give more choices to application developers. The behavior of Linux OS devices has the following features.

- The MIDlet lifecycle mechanism in all Motorola MOTOMAGX phones is the same, although the AMS menu may be different MOTOMAGX.
- The KVM process keeps running in the background and there is no way to suspend it. There is no suspend function built into the AMS.
- Any event that makes the MIDlet get/lose focus will trigger the `pauseApp()` and `startApp()` methods. On MOTOMAGX devices, the Flip behavior cannot currently be ignored and will have an effect on MIDlet focus. This behavior will be added to the MOTOMAGX devices in the future.
- On MOTOMAGX devices that are in the paused state, only the camera is forced to be released.
- The sound device will be taken away from the MIDlet if the system requires its use. For example, with an incoming call, the MIDlet will lose the sound device. When audio resources are required to run in background execution mode, the MIDlet should listen for media player events such as `DEVICE_AVAILABLE/DEVICE_UNAVAILABLE`. The background MIDlet could share the network connection, RMS, and file system with the front end applications, but the front end application has a higher priority.
- System resources can be influenced by work being done by the background MIDlet. This is true if the MIDlet executing in the background has a lot of functionality (such as reading/writing to/from the file system and 3D graphical processing). But if the background MIDlet stops/suspends such activity, then the device will be permitted to enter IDLE mode, thus freeing those system resources.
- A MIDlet must programmatically accommodate any necessary MIDlet pause state. This is especially true for game MIDlets.
- Wherever possible, shared resources should be properly handled before a MIDlet is placed into background mode.
- Care should be taken when releasing and reallocating resources and objects in the lifecycle methods (`pauseApp()` and `startApp()` and the `showNotify()` and `hideNotify()` methods to avoid conflicts and unwanted object creations/initializations.

Chapter 4: MIDP Security Model

Introduction

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

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

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

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

Other benefits to signing a MIDlet for the market are:

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

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

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

The MIDP 2.0 security environment

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

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

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

The Motorola security model contains two types of API:

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

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

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

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

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

MIDP trust

Trust is granted to a MIDlet under the following conditions:

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

Signing authority trust

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

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

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

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

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

Consumer consent trust

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

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

Motorola's general security policy

By default, Motorola handsets trust the following signing authorities:

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

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

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

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

USIllinois=Motorola Manufacturing or TTP Root Certificates.

USUnified = Unified Testing Initiative Root (Java Verified)

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

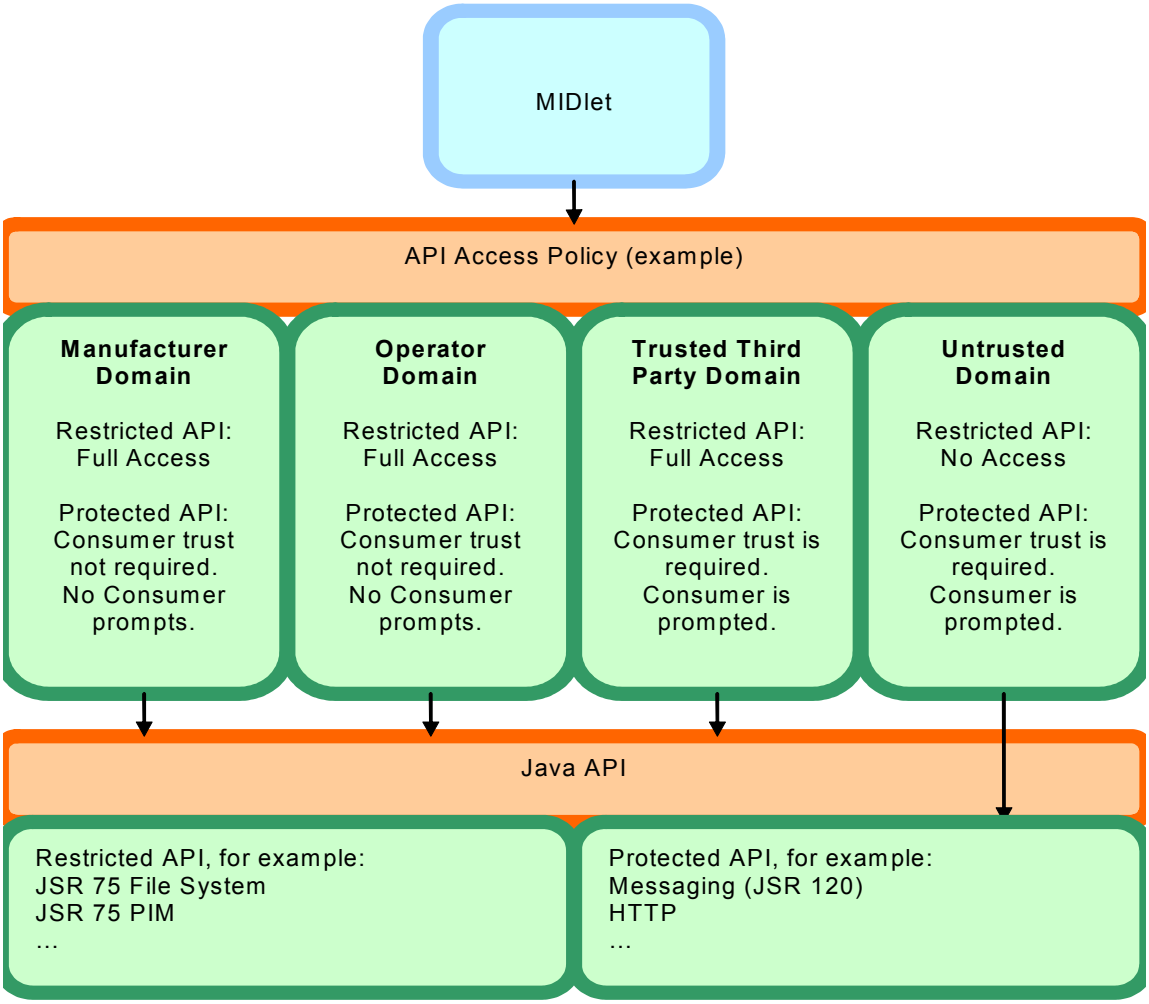


Figure 5: Example Domain and API Policy Model

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

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

Table 1: API Access Matrix

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

Table 1: API Access Matrix (Continued)

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

API access — consumer prompts

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

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

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

Here is a sample scenario:

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

Thie result of this scenario is a consumer prompt giving the choice of “yes” or “no.”

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

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

Operator branding

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

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

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

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

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

Identifying installed Java ME root certificates

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

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

Digital signing and MIDlet development lifecycle

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

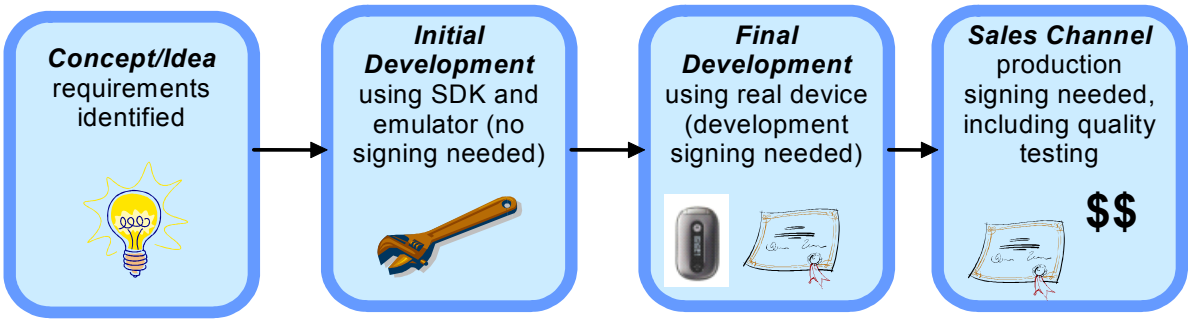


Figure 6: Digital Signing During MIDlet Development

MIDlet development is best done in stages:

1. Concept and specification writing
2. Initial development using a device emulator for testing code
3. Final testing on a real handset
4. Project completion, and acceptance
5. Application delivery to a sales channel

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

On-device testing

You can do much of the development of a MIDlet requiring access to restricted APIs using a Motorola emulator. This is because the APIs are deliberately open and available to the MIDlet in the emulated environment. When the MIDlet is near completion, you will want to test the MIDlet on a device to ensure

that it will work correctly. However, untrusted MIDlets are blocked from accessing the required API on a handset, and a SecurityException is raised if an untrusted attempt is made.

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

Production signing

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

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

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

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

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

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

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

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

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

Development certificates

Once a MIDlet has been developed as far as possible using the Motorola SDK and emulator, it needs testing on an actual handset to ensure correct functionality and behavior. If the MIDlet accesses sensitive

functionality, the MIDlet would have to be trusted to access restricted APIs. This trust is achieved by way of a development certificate.

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

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

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

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

Bound certificates

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

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

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

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

This means that:

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

UID extraction

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

One example of such a need is the internal Motorola process to request a certificate to sign a Java ME MIDlet suite. In order to do this, you need the UID of the handset. To access the internal configurations of

a handset, it is necessary to connect to the handset via USB. Motorola provides a USB driver package for their handsets, which may be downloaded from the MOTODEV web site.

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

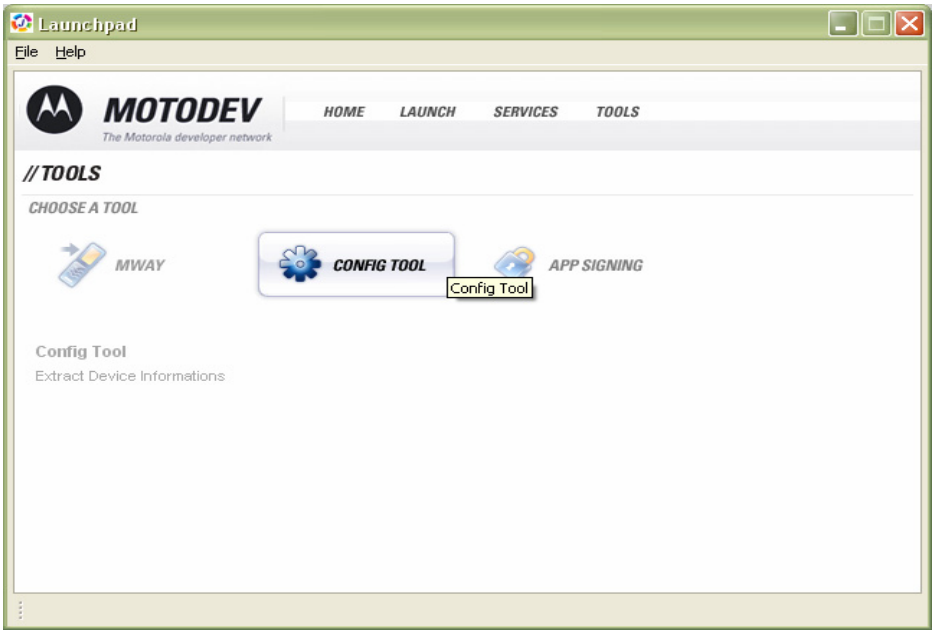


Figure 7: UID Extraction using Motorola Config Tool

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

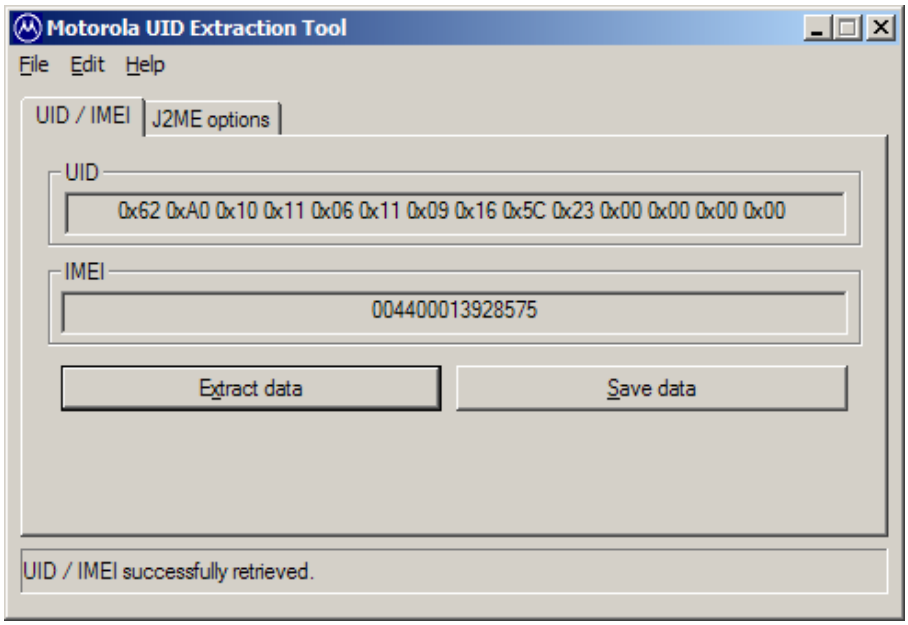


Figure 8: UID extraction using the Motorola UID Extraction Tool

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

Obtaining a development certificate from Motorola

Step 1: Apply for a development certificate

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

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

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

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

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

Step 2: Order a bound certificate

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

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

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

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

Step 3: Install the bound certificate

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

Production signing (MIDlet signing)

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

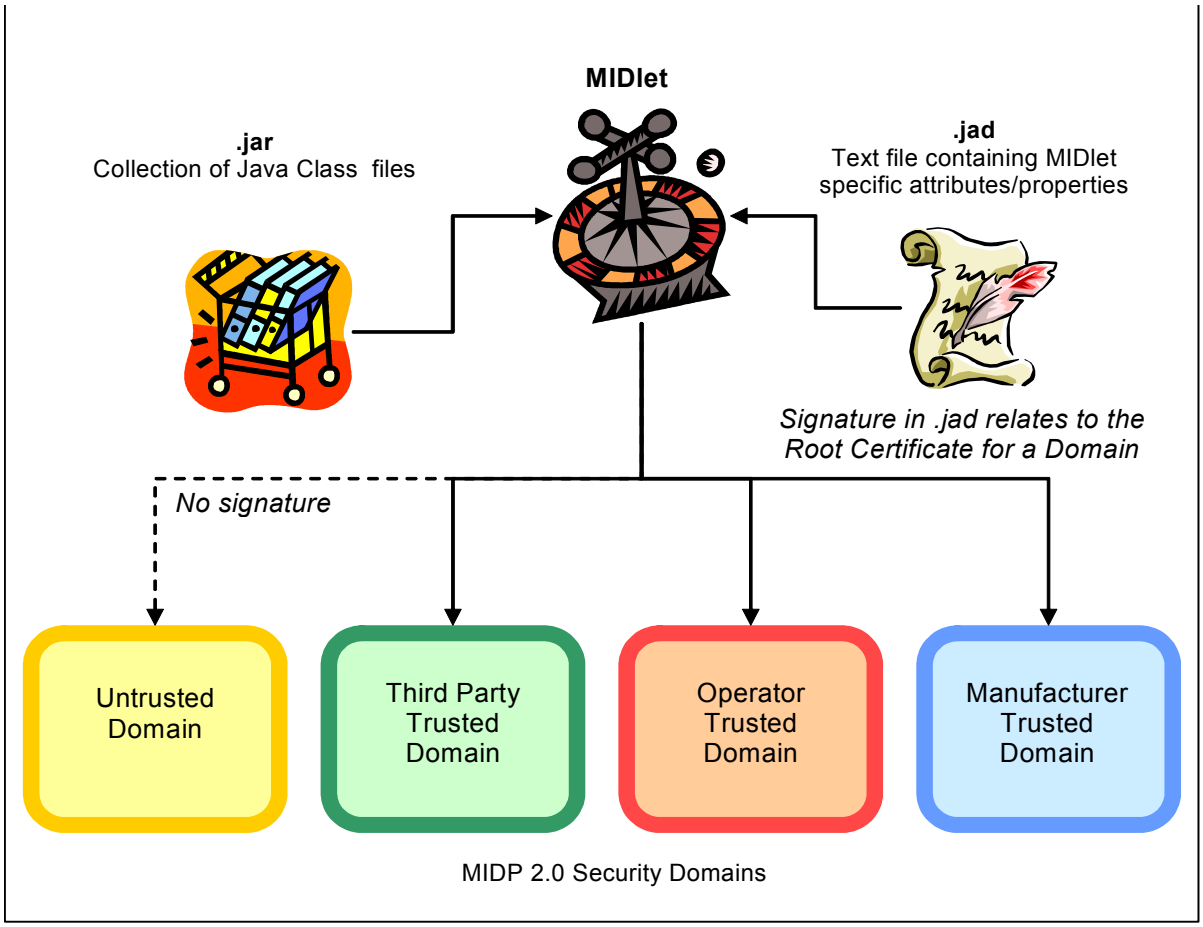


Figure 9: A signature relates a MIDlet to a MIDP 2.0 Security Domain

Choosing a signing authority

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

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

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

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

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

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

Table 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 — summation

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

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

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

Motorola production code signing

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

Motorola security configuration

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

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

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

Table 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
ConnectivityOptions javax.microedition.io.Connector.comm com.vodafone.io.Remotecontrol javax.microedition.io.Connector.bluetooth.client javax.microedition.io.Connector.bluetooth.server javax.microedition.io.Connector.obex.client javax.microedition.io.Connector.obex.servr	blanket (session) oneshot Note: com.vodafone.io is not allowed	blanket (session) Note: com.vodafone.io is not allowed	Allow Note: com.vodafone.io.Remotecontrol is not allowed	Allow

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

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

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

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

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

Summary

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

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

Chapter 5: Network APIs

Network connections

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

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

Table 3 lists the Network API feature/class support for MIDP 2.0:

Table 3: 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

Code Sample shows the implementation of Socket Connection:Socket Connection

Code Sample 2: 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 green icon in the upper left corner) appears on the handset's status bar (Figure 10).

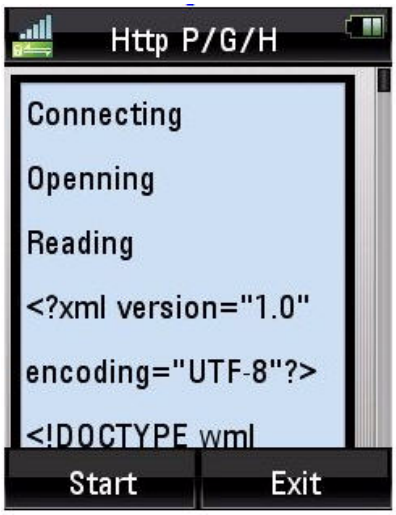


Figure 10: Network Connections Example

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 a USB logical serial port, “USB1”. 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 3 shows the implementation of HTTPS

Code Sample 3: HTTPS

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

  try {
```

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

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

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

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

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

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

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

    }
       }
     }
  }
```

DNS IP

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

Network access

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

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

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

Push registry

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

Mechanisms for push

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

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

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

SMS-Cell Broadcast (SMS-CB) is a messaging service that allows a single originator to broadcast to many in a predefined area. These messages are directed to radio cells rather than to a specific terminal. CBS is an unconfirmed push message which means the originator of the message does not get receipt confirmation. One such application is to broadcast emergency alerts such as severe weather warnings. Support for this feature is both carrier and handset specific.

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 4 is an example of a Push Registry implementation.

Code Sample 4: Push Registry

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

public class MyPushTest extends MIDlet implements CommandListener{

    public Display display;

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

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

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

    public static Command mc;
    public static Displayable ms;

    public MyPushTest() {
```

```
    regConnection = new TextField("Connection port:", "1000", 32,
         TextField.PHONENUMBER);
    regFilter = new TextField("Filter:", "*", 32, TextField.ANY);

    display = Display.getDisplay(this);

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

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

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

    regForm.append(regConnection);
    regForm.append(regFilter);
    regForm.addCommand(regCommand);
    regForm.addCommand(backCommand);
    regForm.setCommandListener(this);

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

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

    public void pauseApp(){}

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

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

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

    public void commandAction(Command c, Displayable s) {
      if((c == unregCommand) && (s == mainForm)) {
        mc = c; ms = s;
        new runThread().start();
      }
      if((c == regCommand) && (s == mainForm)) {
        display.setCurrent(regForm);
      }
      if((c == regCommand) && (s == regForm)) {
        mc = c;
        ms = s;
        new runThread().start();
```

```
  }
  if((c == unregCommand) && (s == unregForm)) {
    mc = c;
    ms = s;
    new runThread().start();
  }
  if((c == backCommand) && (s == unregForm )) {
    display.setCurrent(mainForm); }
  if((c == backCommand) && (s == regForm )) {
    display.setCurrent(mainForm);
  }
  if((c == backCommand) && (s == messageForm)) {
    display.setCurrent(mainForm);
  }
  if((c == exitCommand) && (s == mainForm)) {
    destroyApp(false);
  }
}

public class runThread extends Thread{
  public void run(){
    if((mc == unregCommand) && (ms == mainForm)){
        try{
          registeredConns = PushRegistry.listConnections(false);
       if(unregForm.size() > 0) unregForm.delete(0);
       registeredConnsCG = new ChoiceGroup("Connections", ChoiceGroup.MULTIPLE,
     registeredConns, null);

       if(registeredConnsCG.size() > 0)
                            unregForm.append(registeredConnsCG);
       else unregForm.append("No registered connections found.");

       display.setCurrent(unregForm);
     } catch (Exception e) {
       showMessage("Unexpected " + e.toString() + ": " + e.getMessage());
     }
    }

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

    if((mc == unregCommand) && (ms == unregForm)) {
      try {
        if(registeredConnsCG.size() > 0) {
                            for (int i=0; i<registeredConnsCG.size(); i++) {
                                        if (registeredConnsCG.isSelected(i)) {

      PushRegistry.unregisterConnection(registeredConnsCG.getString(i));

      registeredConnsCG.delete(i);

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

      unregForm.delete(0);

      unregForm.append("No registered connections found.");
                                                                  }
                                        }
                            }
        }
```

```
            } catch (Exception e) { showMessage("Unexpected " + e.toString() + ": " +
            e.getMessage()); }
          }

        }
    }
}
```

Code Sample 5: WakeUp.java

```
// WakeUp.java

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

public class WakeUp extends MIDlet implements CommandListener {

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

    public void startApp() {
      display = Display.getDisplay(this);

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

      display.setCurrent(mainForm);
    }

    public void pauseApp() { }

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

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

      if(c == registerCommand){
        new regThread().start();
      }
    }

    public class regThread extends Thread {
      public void run(){
        try { long delay = Integer.parseInt(tf.getString()) * 1000;
          long curTime = (new Date()).getTime();
          System.out.println(curTime + delay);
          PushRegistry.registerAlarm("WakeUp", curTime + delay); mainForm.append("Alarm
            registered successfully");
        } catch (NumberFormatException nfe) {
```

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

Code Sample 6: SMSSend.java

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

public class SmsSend extends MIDlet implements CommandListener{

    public Display display;

    public static Form messageForm;
    public static Form mainForm;

    public static Command exitCommand;
    public static Command backCommand;
    public static Command sendCommand;

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

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

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

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

    public SmsSend() {
      address_tf = new TextField("Address:", "", 32, TextField.PHONENUMBER);
      port_tf = new TextField("Port:", "1000", 32, TextField.PHONENUMBER);
      message_text_tf = new TextField("Message text:", "test message", 160,
           TextField.ANY); binary_cg = new ChoiceGroup(null, Choice.MULTIPLE,
           binary_str, null);
      display = Display.getDisplay(this);
      messageForm = new Form("SMS_send");
      mainForm = new Form("SMS_send");
      exitCommand = new Command("Exit", Command.EXIT, 0); backCommand = new
           Command("Back", Command.BACK, 0); sendCommand = new Command("Send",
           Command.ITEM, 1);
      mainForm.append(address_tf); mainForm.append(port_tf);
           mainForm.append(message_text_tf); mainForm.append(binary_cg);
           mainForm.addCommand(exitCommand); mainForm.addCommand(sendCommand);
           mainForm.setCommandListener(this);
      messageForm.addCommand(backCommand);
      messageForm.setCommandListener(this);
    }

    public void pauseApp(){ }

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

```
    }

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

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

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

    // inner class?
    public class send_thread extends Thread {
      public void run(){
        try {
          conn = (MessageConnection) Connector.open(address);
          if(!binary_cg.isSelected(0)) {
            txt_message = (TextMessage)
            conn.newMessage(MessageConnection.TEXT_MESSAGE);
            txt_message.setPayloadText(text);
            conn.send(txt_message);
          } else {
            bin_message = (BinaryMessage)
            conn.newMessage(MessageConnection.BINARY_MESSAGE);
            bin_message.setPayloadData(binary_data);
            conn.send(bin_message);
          }
          conn.close();
          showMessage("Message sent");
        } catch (Throwable t) {
          showMessage("Unexpected " + t.toString() + ": " + t.getMessage());
        }
      }
    }

// end SmsSend
}
```

Push message delivery

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

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

- No MIDlet suites running—if no MIDlets are running, the user is notified of the incoming push message and is given the option to run the intended application (Figure 11).

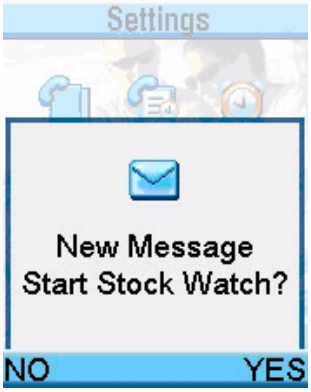


Figure 11: Run Intended Application Query

Table 4: Push Registry Delivery

Push registry with Alarm/Wake-up time for application	Push registry supports one outstanding wake-up time per MIDlet in the current suite. An application uses the TimerTask notification of time-based events while the application is running.
Another MIDlet suite is running during an incoming push	if another MIDlet is running, the user is presented with an option to launch the application that had registered for the push message. If the user selects the launch, the current MIDlet is terminated.
Stacked push messages	it is possible for the handset to receive multiple push messages at one time while the user is running a MIDlet. The user is given the option to allow the MIDlets to end and new MIDlets to begin. The user is given the ability to read the messages in a stacked manner (stack of 3 supported), and if not read, the messages are discarded.
No applications registered for push	if there are no applications registered to handle this event, the incoming push message is ignored.

Deleting an application registered for push

If an application registered in the Push Registry is deleted, the corresponding push entry is deleted, making the port number available for future push registrations.

Security for push registry

Push Registry is protected by the security framework. The MIDlet registered for the push should have the necessary permissions. Details on permissions are outlined in JSR-118 dealing with MIDP 2.0 and security issues.

Chapter 6: Platform Request API

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

For MIDP 2.0, the `javax.microedition.midlet.MIDlet.platformRequest()` method is called and used when the MIDlet is destroyed.The MIDlet is not always required to exit.

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 may continue running in the background. If this API returns true, the MIDlet suite MUST first exit before the content can be fetched. If it cannot, (upon exiting the requesting MIDlet suite) the handset brings the browser to the foreground with the specified URL.

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.

Chapter 7: 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, 4 MB is the maximum RMS space allowed.

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

Interfaces

- RecordComparator
- RecordEnumeration
- RecordFilter
- RecordListener

Classes

- RecordStore

Exceptions

- InvalidRecordIDException
- RecordStoreException
- RecordStoreFullException
- RecordStoreNotFoundException
- RecordStoreNotOpenException

Chapter 8: 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.

Chapter 9: JSR-30 CLDC 1.0

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

Chapter 10: JSR-75 PDA Optional Packages

PIM API

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

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

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

FileConnection API

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

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

Chapter 11: JSR-82 - Bluetooth API

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

On handsets supporting Bluetooth, Motorola supports both the `javax.bluetooth` package and the `javax.obex` package. Protocols “tcpobex” and “btgoep” are supported in the `javax.obex` package.

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

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

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

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

For more information about permissions, certificates and signing processes for a MIDlet, how Motorola handsets deal with the MIDP 2.0 security policy, and the benefits of having a signed MIDlet, refer to Chapter 3:MIDP Security Model. You may also want to access the MIDlet Testing and Signing section on the MOTODEV portal (https://developer.motorola.com/techresources/testingcertification/). For details on Motorola's implementation of MIDP 2.0 security, see Chapter 3 of this guide.

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

Chapter 13: JSR-120 - WMA

Wireless Messaging API (WMA)

Motorola has implemented certain features that are defined in the Wireless Messaging API (WMA) 1.0. The complete specification document is defined in JSR-120. The JSR-120 specification states that developers can be provided access to send (MO - mobile originated) and receive (MT - mobile terminated) SMS (Short Message Service) on the target device.

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

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

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

SMS client mode and server mode connection

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

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

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

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

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

SMS port numbers

When the address contains a port number, the TP-User-Data of the SMS contains a User-Data-Header with the application port addressing scheme information element. When the recipient address does not contain a port number, the TP-User-Data does not contain the application port addressing header. The J2ME MIDlet cannot receive this kind of message, but the SMS will be handled in the usual manner for a standard SMS to the device. When a message identifying a port number is sent from a server type `MessageConnection`, the originating port number in the message is set to the port number of the `MessageConnection`. This allows the recipient to send a response to the message that will be received by this `MessageConnection`. However, when a client type `MessageConnection` is used for sending a message with a port number, the originating port number is set to an implementation specific value and any possible messages received to this port number are not delivered to the `MessageConnection`. For more information refer to sections A.4.0 and A.6.0 of JSR-120.

When a MIDlet in server mode requests a port number (identifier) to use and it is the first MIDlet to request this identifier it will be allocated. If other applications apply for the same identifier then an `IOException` will be thrown when an attempt to open `MessageConnection` is made. If a system application is using this identifier, the MIDlet will not be allocated the identifier. The port numbers allowed for this request are restricted to SMS messages. In addition, a MIDlet is not allowed to send messages to certain restricted ports; If this is attempted, a `SecurityException` is thrown. JSR-120 Section A.6.0 Restricted Ports: 2805, 2923, 2948, 2949, 5502, 5503, 5508, 5511, 5512, 9200, 9201, 9203, 9207, 49996, 49999. If you intend to use SMSC numbers, review A.3.0 in the JSR-120 specification. A MIDlet uses an SMSC to determine the recipient number.

SMS storing and deleting received messages

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

SMS message types

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

SMS message structure

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

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

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

SMS notification

Some examples of SMS interaction with a MIDlet are:

- A MIDlet handles an incoming SMS message if the MIDlet is running and registered to receive messages on the port (identifier).
- When a MIDlet that is registered to receive messages on the port number of the incoming message pauses, the user is queried to launch the MIDlet.
- If the MIDlet is not running and the Java Virtual Machine is not initialized, then a Push Registry will be used to initialize the Virtual Machine and launch the J2ME MIDlet. This only applies to trusted, signed MIDlets.
- If a message is received and the untrusted unsigned application and the KVM are not running then the message will be discarded.
- There is a SMS Access setting in the Java Settings menu option on the handset that allows the user to specify when and how often to ask for authorization.

Before the connection is made from the MIDlet, the options available are:

- Always ask for user authorization
- Ask once per application
- Never Ask

Table 5 lists Messaging features/classes supported in the device.

Table 5: 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 7 shows the implementation of the JSR-120 Wireless Messaging API

Code Sample 7: 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 8: JSR-120 Wireless Messaging API Sample1.java

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

```
import java.io.*;

public class JSR120Sample1  {

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


    public void handleMessages() {

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

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

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

      // send message (by invoking a send method)

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

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


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

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

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

}
```

Chapter 14: JSR-135 - Mobile Media API

Network connections

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

- Tone Sequence
- Sampled Audio
- MIDI

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

Code Sample 9 is an example of the usage of the JSR-135 Mobile Media API:

Code Sample 9: JSR-135 Mobile Media API

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


public class MyMP3Player {

    Player player;

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


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


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

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


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

}
```

ToneControl

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

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

The available method for ToneControl is:

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

VolumeControl

VolumeControl is an interface for manipulating the audio volume of a Player.

The JSR-135 Mobile Media API implements the public interface, VolumeControl. VolumeControl settings are:

Volume Settings	specifies the output volume using an integer value between 0 and 100.
Specifying Volume in the Level Scale	specifies volume in a linear scale. It ranges from 0 - 100, where 0 represents silence and 100 represents the highest volume available.
Mute	setting mute on or off does not change the volume level returned by `getLevel`. If mute is on, the Player doesn't produce an audio signal. If mute is off, the player produces an audio signal and the volume is restored.

Available methods for `VolumeControl`:

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

StopTimeControl

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

Available methods for `StopTimeControl`:

`getStopTime()`	Gets the last value successfully by `setStopTime`.
`setStopTime(long stopTime)`	Sets the media time at which you want the Player to stop.

Manager class

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

Supported multimedia file types

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

Image media

Table 6: Image Media

File Type	Codec	Functionality
JPEG	JPEG	Capture

Table 7: Image Media

File Type	Functionality
GIF 87a, 89a	Playback

Table 8: Image media types and descriptions

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

Audio media

Table 9: 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
MP3	MPEG-1 layer III
AMR	AMR
BAS	General MIDI

Table 10: Audio MIME types

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

Video media

Table 11: 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

Table 12 security policy is set per operator requirements when the handset is shipped.

Table 12: Security Policy

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

Table 12: Security Policy

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

Permissions

Table 13 lists individual permissions in the MultimediaRecord function group.

Table 13: 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.

Chapter 15: JSR-139 - CLDC 1.1

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

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

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

Chapter 16: JSR-172 - Web Services API

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

The web services API contains two optional packages:

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

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

For Motorola-specific system properties, see “System Properties” on page 135.

Chapter 17: JSR-177- SATSA and Crypto

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

SATSA-APDU optional package

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

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

Crypto API

JSR-177 Crypto is a subset of the Java™ SE cryptography API. It provides cryptographic functions to support encryption/decryption, message digest and signature authentication.

JSR 177 Crypto is composed of three main classes:

- MessageDigest (hash): Used to guarantee the physical integrity of received data.
- Signature: Used for verification of digital signature.
- Cipher: Used for encryption and decryption.

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

Chapter 18: JSR-179 Location API

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

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

API requirements

Security

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

Motorola-specific implementation

Location

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

ProximityListeners

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

Landmark

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

AddressInfo

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

Orientation

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

LandmarkStore

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

Chapter 19: JSR-184 - Mobile 3D Graphics

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

Applications for this API include:

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

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

Chapter 20: JSR-185 - JTWI

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

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

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

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

Chapter 21: JSR-205 - WMA 2.0

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

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

Chapter 22: JSR-211 - Content Handler

The Content Handler API enables an application to use URL, content type or content handler ID to invoke registered J2ME and non-Java applications. This method also enable actions to be executed by the handler on the invoked content.

On Motorola devices, the implementation of the Content Handler API is based on JSR-211 from the Java™ Specification Request. Details of the specification are available on the Java Community Process (JCP) website: http://jcp.org/en/jsr/detail?id=211.

Chapter 23: JSR-226 - Scalable 2D Vector Graphics API

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

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

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

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

Chapter 24: Motorola Get URL

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

Flexible URL for downloading functionality

The following rules apply:

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

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

Security policy

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

Chapter 25: Motorola Secondary Display API

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

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

User interface restrictions

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

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

Flip-open/Flip-close event handling

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

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

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

Exception handling

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

Push enabled applications

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

Feature interaction

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

Security

The Secondary Display API follows the MIDP security model.

Chapter 26: Motorola CMCC Enhancements API

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

- User Interface, which implements the scale package
- Phonebook

User interface

Overview

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

Package

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

Interface/class implementation

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

ScaleImage

drawScaledRegion

•public static void drawscaledRegion throws ScaleImageException

•IllegalArgumentException, NotSupportScaleReq

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

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

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

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

Phonebook

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

Overview

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

Package

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

Interface/class implementation

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

- PhoneBookEntry
- PhoneBook

Definition of class

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

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

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

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

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

Table 14: PhoneBookEntry Field Summary

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

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

Table 15: PhoneBook Field Summary

Method Summary:

Table 16:

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

Table 17: PhoneBook Method Summary

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

Chapter 27: Motorola Multiple APN Support

Access Point Name (APN) support is a feature included in the Linux-Java (MOTOMAGX) platform, version 6.3. Supporting multiple APNs provides a simple way for Java developers to link the application to an existing profile. You do not need to select the profile and can connect to the network directly.

Specifying a network profile

The JAD `MOT-MIDlet-Web-Session` attribute must be a profile name. During download, the installation checks for the existence of the profile. If the profile does not exist, the installation fails.

Displaying the network setting

When viewing the MIDlet property, the following logic is used to determine which profile to display.

1. If the JAD specifies an existing profile, the profile name is displayed. This profile selection cannot be changed.
2. If the JAD specifies the attribute but the named profile does not exist, or, if the JAD attribute is not specified, the user may select the profile from the available profiles.

Selecting the network profile

When running a Java suite that has been installed with the `MOT-MIDlet-Web-Session` attribute, the network profile should be selected in the following order:

1. If the `MOT-MIDlet-Web-Session` attribute specified profile exists, then this profile will be used to connect to the network.
2. If the user selects an existing profile, then the selected profile is used.
3. If a Java session profile exists, then it is used.
4. If the default Operator-specific Java profile can be automatically provisioned, then the default Operator-specific Java profile is used.
5. If only one profile exists, then that one profile is used.
6. Display the connection profile selection dialog, and the user selects the profile.

If a Java suite has been installed without a specified profile in the JAD, a network profile should be selected in the following order:

1. If the user selects an existing profile, then the selected profile is used.
2. If a Java session profile exists, then it is used.
3. If the default Operator-specific Java profile can be automatically provisioned, then the default Operator-specific Java profile is used.
4. If only one profile exists, then that profile is used.
5. Display the connection profile selection dialog, and the user selects the profile.

Chapter 28: Motorola ModeShift Technology

The Motorola ModeShift feature allows users to have different key pads based on different technology modes. A mode defines the keys that are available to the user, based on the active application. Table 1 lists all possible modes for a Motorola device.

NOTE: Not all modes are available on all handsets. Check the specifications sheet for any given handset to find out which modes are supported.

Table 18: Possible ModeShift technoloty modes

Mode	Description
MODE_STANDBY	Standby mode. None of the keys are active.
MODE_NAVIGATION	Navigation mode. Some navigation keys and the FastScroll Wheel are active.
MODE_PHONE	Phone mode. All numeric and some navigation and FastScroll Wheel keys are active.
MODE_MUSIC	Basic music mode. All music keys and some navigation and FastScroll Wheel keys are active.
MODE_STILLCAPTURE	Still capture mode. Some navigation, all zoom (Zoom in and Zoom out), Capture/Playback Toggle, and Still/Video Toggle keys are active.
MODE_VIDEOCAPTURE	In this mode the device Video capture mode. Some navigation, all zoom (Zoom in and Zoom out), Capture/Playback Toggle, and Still/Video Toggle keys are active.
MODE_STILLPLAYBACK	Still playback mode. Some navigation, FastScroll Wheel, and Music and Clear keys (Music Jump and Clear/Back) are active. Note that this mode isn't available for some devices.
MODE_VIDEOPLAYBACK	Video playback mode. Some navigation keys and Capture/Playback Toggle keys are active.

Figure 12: STILLCAPTURE, STILLPLAYBACK, VIDEOCAPTURE, and VIDEOPLAYBACK

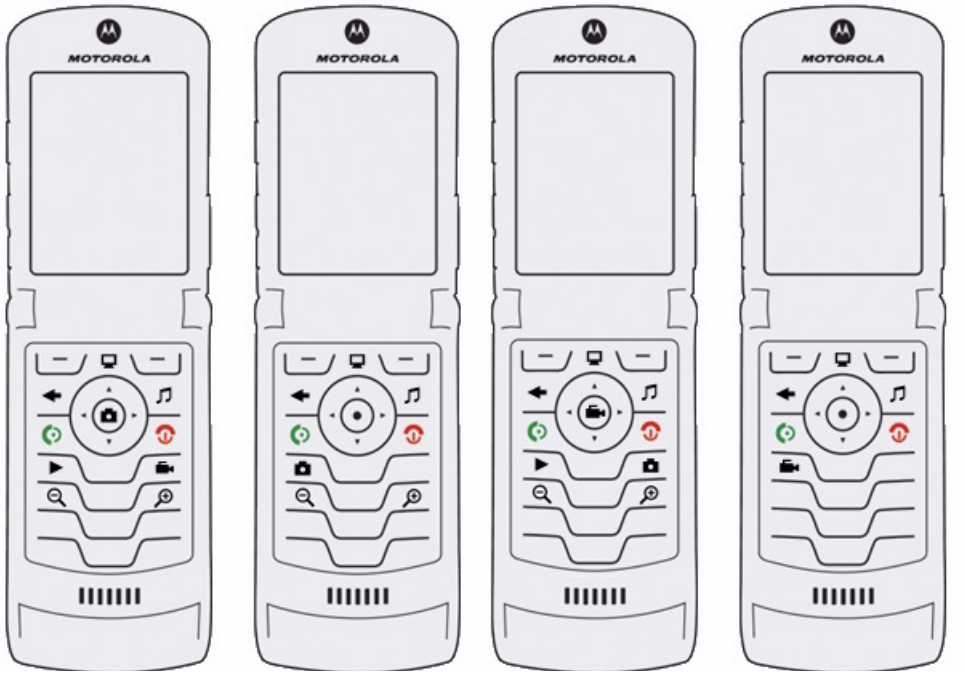
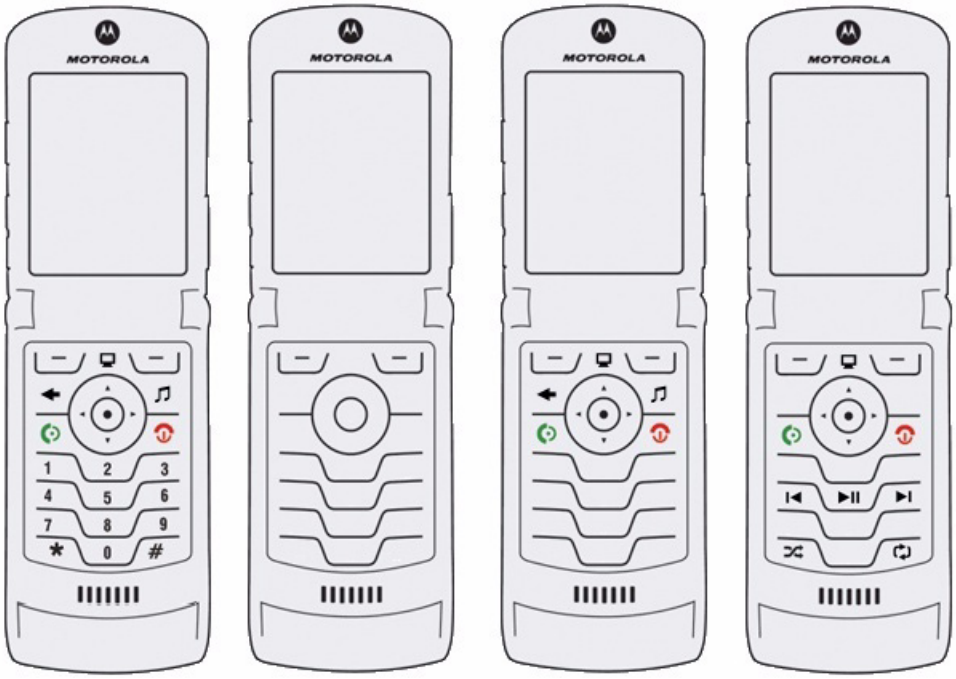


Figure 13: MODE_PHONE, MODE_STANDBY, MODE_NAVIGATION, and MODE_MUSIC



The following table lists the non-standard key codes available with the ModeShift Technology.

Table 19: ModeShift non-standard key codes

Key Group	Keys	Key Codes	Description
FastScroll Wheel	Navigation wheel	See the FastScroll Wheel documentation.	See the FastScroll Wheel documentation.
Music and Clear	Music Jump Key Clear/Back	0x1048 0xffffffff8	Jump key. Launches media finder. Clear/Back
Music	Music Play/Pause Skip Forward Skip Back Shuffle Repeat	0x1049 0x104c 0x104b 0x4002 0x4003	Play/Pause Next Previous Music shuffle Music repeat
Zoom	Zoom in Zoom out	0x4005 0x4004	Zoom in Zoom out
Capture/Playback Toggle	Capture/Playback Toggle	0x4006	Shift between capture and playback modes
Still/Video Toggle	Still/Video Toggle	0x4007	Shift between still and video modes

Setting ModeShift technology modes

A Java ME application can select the mode either statically, by providing the attribute in the JAD file, or at run time, by invoking a call to `MIDlet.platformRequest()` as descrubed next.

- Adding and attribute in the JAD file. In this case the Morphong-Mode attribute in the JAD file can be used to specify the mode setting attribute `Morphing-Mode` followed by the mode. For example: `Morphing-Mode:MODE_INT_PHONE`.
- Using platformRequest: The `MIDlet.platformRequest(String URL)` method from the Java ME application sets the ModeShift technology mode at run time. The URL string is `Morphing://MODE_XXXX`.

ModeShift system properties strings

Two system properties are used to acquire information related to the supported ModeShift states:

- System property "`Morphing.current`" returns the device's current mode

- System property "`Morphing.list`" returns a stringwith a list of modes supported by the device.

Both properties are retrieved using the `System.getProperty()` method. The sample code shown in sample 1 specifies this method.

Code Sample 10: Retrieving ModeShift technology modes

```
// Retrieving the current ModeShift mode.
String currentMode = System.getProperty ("morphing.current");

// To see the currentMode value use the System.out.println() method.
// The output will be one of the ModeShift technology modes listed in the 'Possible
          ModeShift technology modes' table.
```

Selecting the initial MIDlet ModeShift technology mode

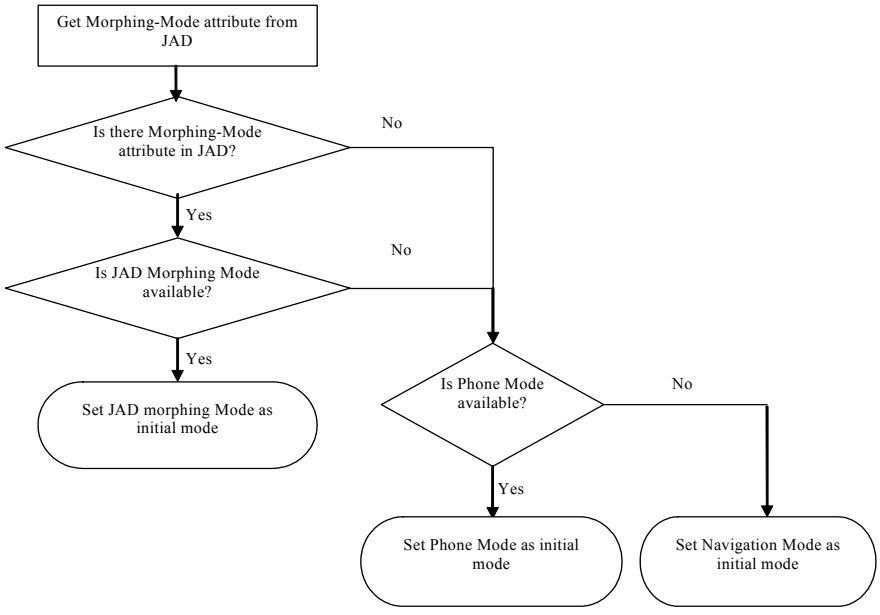
The initial ModeShift technology mode for a MIDlet is set by adding the mode into the MIDlet JAD file. To specify the JAD attribute, include the `ModeShift-Mode` exactly as specified.

Code Sample 11: JAD ModeShift attribute

```
// JAD file attribute
ModeShift-Mode:MODE_INT_MUSIC
```

The flowchart shown in Figure 3 illustrates how the system set an initial ModeShift technology mode from the device's JAD attribute.

Figure 14: Initial MIDlet ModeShift technology mode selection



NOTE: If the device system doesn't support the initial ModeShift technology mode, it checks to see if `Phone-Mode` is present. If it is not present, it select the `Navigation Mode` as the initial mode.

Background MIDlet

Whan a MIDlet running in the background sets the ModeShift technology mode, the mode of the handset is not changed until the MIDlet is no longer running in the foreground.

Chapter 29: Fast Scroll Wheel

Some of the newer LJ handsets come with an input device called a “FastScroll Wheel” which is optimized for navigation. It is used to navigate large amounts of data efficiently and intuitively. Some examples of this application are navigating music lists or phonebook entries.

The FastScroll Wheel is supported by the KJava Virtual Machine (KVM) as well. This provides several more convenient operations for a Java ME application. This chapter briefly describes the suage of the Fast Scroll Wheel with various LCDUI classes. Some of these features, like key codes, may be specific to one particular handset, but others may be relevant to other handsets supporting this navigation method.

Relevant LCDUI class support

javax.microedition.lcdui.Form
When the Fast Scroll wheel is scrolled clockwise, the form moves focus to the next focusable item on the form. When the FastScroll wheel is scrolled counter-clockwise, Form moves focus to the previous focusable item. If the user taps or presses the right tapping zone, Form scrolls to the next page and moves focus to the first item with content on that page. If the user taps or presses the left tapping zone, Form scrolls to the previous page and moves focus to the first item with content on that previous page.

javax.microedition.lcdui.Alert
When the Fast Scroll Wheel is scrolled clockwise, Alert’s content scrolls backwards. When the wheel is scrolled counter-clockwise, Alert’s content scrolls forward. If the user taps or presses the right tapping zone, Alert scrolls to the next page. If the user taps or presses the left tapping zone, Alert scrolls to the previous page.

javax.microedition.lcdui.List
When the Fast Scroll Wheel is scrolled clockwise, List moves focus to the next element. When the wheel is scrolled counter-clockwise, List moves focus to the previous element.. If the user taps or presses the right tapping zone, List scrolls to the next page and moves focus to the first element on that page. If the user taps or presses the left tapping zone, List scrolls to the previous page and moves focus to the first element on that page.

javax.microedition.lcdui.DataField
If a DataField is selected and the user scrolls the wheel clockwise, the value of the current section will decrease continuously. If the user scrolls the wheel counter-clockwise, the value of the current section will increase continuously. If a DataField is selected, and the user taps or presses the right tapping zone, the value of the current section will be decreased by one. If the user taps or presses the left tapping zone, the value of the current section will be increased by one.

javax.microedition.lcdui.guage (Interactive)
When a guage is highlighted and is in a selected state, scrolling the wheel clockwise causes the KVM to increase the value of the Gauge by one step. If the user scrolls the wheel counter-clockwise, KVM decreases the value of the Gauge by one step. If the user presses or taps the right tapping zone, the value

of the selected Guage will be devreased by a chunk and if the user presses the left tapping zone, the value of this Guage will be increased by a chunk.

avax.microedition.lcdui.ChoiceGroup
When the Fast Scroll Wheel is scrolled clockwise, ChoiceGroup moves focus to the next element step by step, and when the last element is reached, the focus is moved to the next item. When the wheel is scrolled counter-clockwise, ChoiceGroup moves focus to the previous element step by step, and when the last element is reached, meaning the first element, the focus moves to the previous item.

javax.microedition.lcdui.Canvas

javax.microedition.lcdui.CustomItem
Although the FastScroll wheel is not a MIDP standard key, KVM is still able to send all wheel key events to Canvas and CustomItem.

For FastScroll wheel control, four new keys and relevent key events are defined. A Java ME developer can use key code to distinguish different keys.

[

Table 4: Fast Scroll Wheel key mapping

Key Description	Key Code	Key Event
Wheel touched	0x4010	Key press / key release
Wheel page up	0x4011	Key press
Wheel page down	0x4012	Key press
Wheel scroll	0x5FFF0000 - 0x5FFF0081	Key press

The Fast Scroll wheel includes a series of Java ME key codes. The range of key codes is 0x5FFF0000 - 0x5FFF0081the scroll direction benchmark is 0x5FFF0040. Key code minus benchmark is the step information.; a postive value means clockwise scrolling, a negative value means counter-clockwise scrolling. A code sample is shown next.

Code Sample 12: Public void keyPressed(int keycode)

```
{
   if (ox5fff0000 <= keycode) & (keycode <= 0x5fff0081)) //this is fastscroll wheel key
```

"Option" menu

The “Options” menu also supports the Fast Scroll wheel. When the “Options” menu is selected, if the user scrolls the wheel clockwise or presses the right tapping zone, the highlighted area will be scrolled up. If the user scrolls the wheel counter-clockwise or presses the left tapping zone, the highlighted area will be scrolled down.

Chapter 30: Motorola Fun Lights API

This feature provides access to various light regions of the handset. The API controls additional characteristics like brightness, colors variation, light intensity and light control (which lights will be on/off). Applications already have some degree of control over handset lights, for example vibrator and backlight control. However, they can't control the lights on specific regions of the handset.

Using Fun Lights, the MIDlet turns the Fun Lights on/off based on the needs of the application. Some examples of these functions are:

- With Karaoke or jukebox applications the variation of the brightness or state of the light (on/off) according to the beats of the music.
- With games, car raceing for example, the developer can use Fun Lights with variations in intensity and colors to warn of potenial collisions.
- Alarm clock or stopwatch applications can use the Fun Lights with features related to the timer.

Fun Lights Regions

Each light source of a handset needs to be identified on the software code, in a way that they are defined by regions. There are different ways to identify a light source, for example numerically or alphabetically.

Motorola handsets designate regions that light up and/or change colors with numbers. For instance, a MIDlet communicates with the native code of the handset to learn the number of light regions and is able to map those regions with numbers.

Region Control

Any application can use the hardware regions. To have the correct use of the regions, the applications need to maintain control of the region. The MIDlet is only able to use a region over which it has control. The flowchart in Figure 15 shows how the MIDlet requires control.

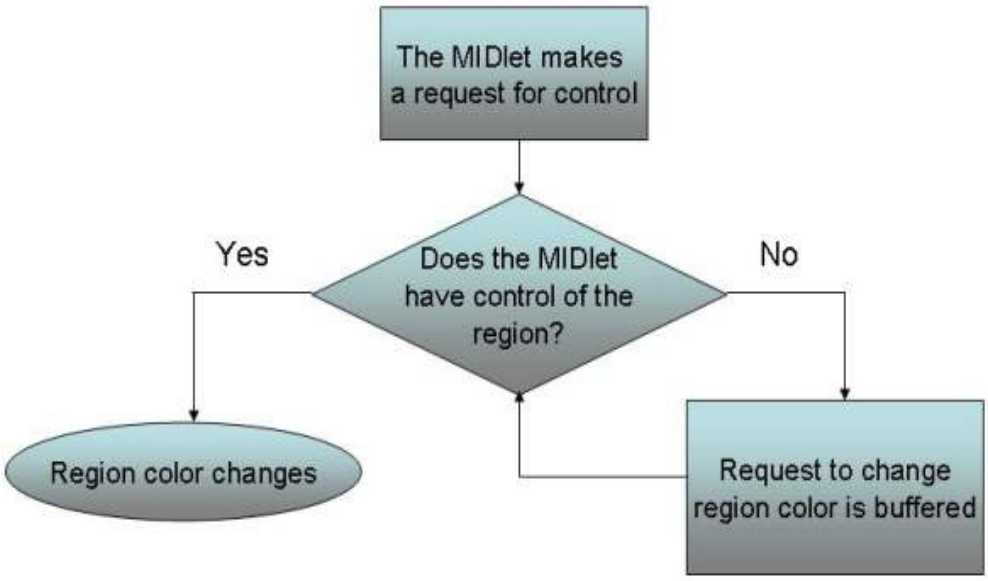


Figure 15: Request for region control

To obtain control of a region use the `getControl()` method. To release control of a region use `releaseControl()` method.

Regions Sets

The regions for MOTORAZR maxx V6 are outlined in the following table.

Table 20: MOTORAZR maxx V6

Region ID	Region Name	Common Description	Color Scheme
1	Primary Display	Main display	Brightness levels
2	Secondary Display	CLI	Brightness levels
5	Keypad	Entire keypad without any segmentation	Signle-color
10	Music Keypad 1	Music keys on the flip	-
15	Privacy	Red LED near the camera	Signle-color
16	Sidebar	2 vertical lines on the sides of the flip	Brightness levels

Figure 16 represents Fun Lights on the MOTORAZR² V9.

.

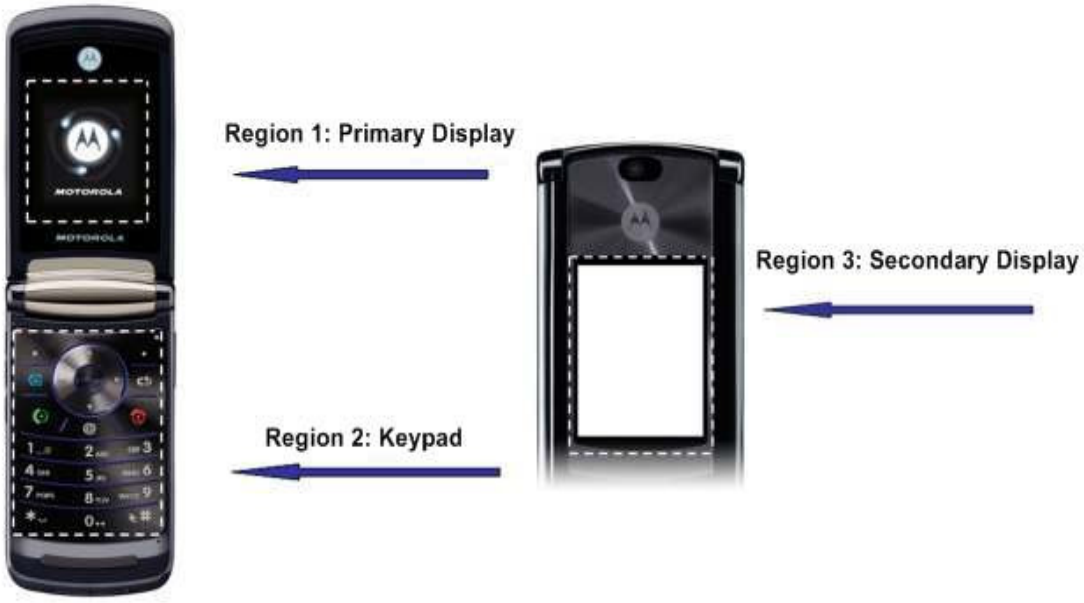


Figure 16: Fun Lignts on the MOTORAZR^2 V9.

Single-color scheme

Single-color region: The light can only be turned on/off.

If any non-zero color value uses the `setColor()` method, the color changes FunLight.WHITE; otherwise the region is turned off. The `getColor()` method returns FunLight.WHITE if a region is turned on or FunLight.BLACK if it is turned off.

Brightness levels scheme

Brightness levels can be set for regions. The example in Table 21 has the change of brightness level for the color 0xRRGGBB.

Table 21: Color brightness conversion

Passed color	Step 1	Step 2
0xRRGGBB	**Formula:** BrLevel = 1/3*(0xRR + 0xBB + 0xGG) **Result:** Brightness level (range: 0 - 255).	Brightness level range: 0 - 255 HW's brightness level: range: 0-15 The scaling procedure algorithm: • A zero brightness level is converted to a zero value level of brightness on the handset. • Non-zero brightness levels (1-255) are divided into 15 equal ranges. Each of these ranges corresponds to one of the brightness levels on the handset. If the division can not be done evenly, the residue is added into the first range.

NOTE: In step 1, the brightness level is calculated from a passed color and step 2, scaling a brightness level (from step #1) to a brightness level that is supported by the device.

Table 22: Brightness levels

Brightness Levels 0x00 - 0xFF	Hardware's Brightness Levels 0 - 15	getColor() returns
0x00	0	0x000000
0x01 - 0x11	1	0x111111
0x12 - 0x22	2	0x222222
0x23 - 0x33	3	0x333333
0x34 - 0x44	4	0x444444
0x45 - 0x55	5	0x555555
0x56 - 0x66	6	0x666666
0x67 - 0x77	7	0x777777
0x78 - 0x88	8	0x888888
0x89 - 0x99	9	0x999999
0x9A - 0xAA	10	0xAAAAAA
0xAB - 0xBB	11	0xBBBBBB
0xBC - 0xCC	12	0xCCCCCC
0xCD - 0xDD	13	0xDDDDDD
0xDE - 0xEE	14	0xEEEEEE
0xEF - 0xFF	15	0xFFFFFF

RGB 12 bits Scheme

The region supports lights of several colors. The color is specified in RGB format (8 bit for each R, G, and B component). When 0x00RrGgBb color is passed using the `setColor()` method, the color converts to 0x00RGB (4 bit for each R, G, B component), which in turn sets the region. The `getColor()` method returns the color in 0x00RRGGBB format (0x00RGB color extends by duplicating the R, G and B components).

The Human Factor

Due to human factor concerns, the refresh rate of the LEDs must be less than 10 Hz. Therefore, the period between changing the light status for a region should be more than 100 milliseconds otherwise a request for changing a light status is ignored. The functions `Region.getControl()` and `Region.setColor()` return FunLight.IGNORED in the following cases due to the human factor:

- The latest update (successful or queued) of the region's control was less than 100 milliseconds ago.
- If a MIDlet has the region's control and the latest update (successful or queued) of the region's color was less than 100 milliseconds ago.

Fun Lights API

This API uses several illumination methods to create visual stimulus, with the objective of increasing the visual interaction and the sensorial perception. There are six primary elements related to the handsets that possess Fun Lights: personality, behavior, light source, transmission, triggers and attributes.

Fun Lights uses a group of patterns that determine the variation of the sections brightness. Based on the activity of the application determined Fun Lights turn on/off. The fun light, also known as Fun Lights Pattern (FLP), controls all the commands that manages the effects, color controls, time and light intensities on all regions.

A Light Pattern

A Light Pattern contains the commands to control lighting attributes.

There are two versions of FLP files:

- FLP1 (Audio-based) - shows lights based on the audio.
- FLP2 (Time-based) - shows lights based on the time. The behavior of the lights is based on the command list that the handset receives explicitly with intensity and if the light should be turned on/off.

NOTE: The implementation of the Fun Lights API was just developed for FLP1 (time-based).

Some groups of light pattern can be found as:

- Ring Lights: a group of user-selectable light patterns that can be activated by a type of incoming call (Video, Voice, PTT, etc.).
- Event Lights: as a group of light patterns that are activated by events other than Incoming calls (Alarm Clock, Power Up and Bluetooth). These patterns are predefined and are not user-selectable.
- Message Lights: a group of user-selectable light patterns which are activated by any type of incoming message (MMS, SMS and Email).

Light Pattern format

A Light Pattern file is a list of commands that tell the device explicitly which light source to turn on/off at varying intensities. A Light Pattern file should be in ASCII coded format.

Figure 17 is an example of a Light Pattern file.

```
Pattern ID:1
Type:FLP
Regions:1
Version:0002
Period(1601);
Set(10,0,0xFFF,0);
Set(1,0,0x0,0);
Set(1,0,0x111,200);
Set(10,0,0xDDD,200);
Set(1,0,0x333,400);
Set(10,0,0xBBB,400);
Set(1,0,0x555,600);
Set(10,0,0x999,600);
Set(1,0,0x777,800);
Set(10,0,0x777,800);
Set(1,0,0x999,1000);
Set(10,0,0x555,1000);
Set(1,0,0xBBB,1200);
Set(10,0,0x333,1200);
Set(1,0,0xDDD,1400);
Set(10,0,0x111,1400);
Set(1,0,0xFFF,1600);
Set(10,0,0x0,1600);
```

Figure 17: Light Pattern file

The following table lists fields that must be mentioned in the beginning of the light pattern file.

Table 23: Header information:

Fields	Description
Type	FLP
Regions	n, where n is the number of regions used.
Version	n, where n is the version number
Pattern ID	n, where n is the index value to a name for the pattern

The functions in the following table tell the handset to adjust a particular LED's status.

Table 24: Set (a, b, c, d)

Functions	Description
a	Represents a region. The value input shall be a positive integer between 1 and 256
b	Represents the region type. The '0' value shall be used.
c	Represents the index value of the color to be used for a particular threshold range. The index value is 12-bit hexadecimal in RRGGBB format.
d	Represents the instance in time when a command should be executed in milliseconds from the start. The start point is 0.

NOTE: `set()` commands should be sorted in increasing order for d (the instance in time) parameter.

Chapter 31: Motorola Bluetooth Remote Control API

This proprietary feature allows the phone to be controlled using a remote Bluetooth device, such as a stereo Bluetooth headset. This API supports the implementation of commands received and the action of events sent by the Bluetooth remote device to the phone.

This API supports events sent by a third party Bluetooth service that control the audio and video buttons. The support informs which instance is active from the class `com.motorola.extentions.RemoteControl`.

Remote Control class, methods, variables and events

This feature implements the class interface needed for remote control device support. The implementation is in accordance with Motorola's proprietary Bluetooth Remote Control API support.

The class `RemoteControl()` implements the following four methods:

- void `gainControl()`: establishes which method receives event notification from the remote control device. Only the instance that has the control can receive event notifications.
- protected void `handleEvent(int event)`: sends notification of event occurance on the remote control device such as button press, gain control, etc. This method iscalled by the platform.
- boolean `hasControl()`: checks if control is available for an instance of RemoteControl. It returns a Boolean value: true if control is available, false if control is not available.
- void `loseControl()`: sends notification that control of the Bluetooth remote control has changed and future events notification will not be sent to this instance.

Variables summary

For each button pressed on the remote control device, an associated event parameter is used on method handleEvent (int event). These parameters are integers with constant values and are presented in the following table:

com.motorola.extensions.RemoteControl

```
public static final int BACKWARD_PRESS 9
public static final int BACKWARD_RELEASE 10
public static final int CONTROL_GAINED 21
public static final int CONTROL_LOST 22
public static final int EXIT_PRESS 25
public static final int EXIT_RELEASE 26
public static final int FAST_FORWARD_PRESS 11
public static final int FAST_FORWARD_RELEASE 12
```

```
public static final int FORWARD_PRESS 7
public static final int FORWARD_RELEASE 8
public static final int MUTE_PRESS 19
public static final int MUTE_RELEASE 20
public static final int PAUSE_PRESS 3
public static final int PAUSE_RELEASE 4
public static final int PLAY_PRESS 1
public static final int PLAY_RELEASE 2
public static final int REWIND_PRESS 13
public static final int REWIND_RELEASE 14
public static final int STOP_PRESS 5
public static final int STOP_RELEASE 6
public static final int VOLUME_DOWN_PRESS 17
public static final int VOLUME_DOWN_RELEASE 18
public static final int VOLUME_UP_PRESS 15
public static final int VOLUME_UP_RELEASE 16
```

Next is a code sample where, after a button is pressed, the phone returns a message to the user displaying which action was performed.

Code Sample 13: Bluetooth Remote Control API code example

```
package com.motorola.moja.test;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Graphics;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import com.motorola.extensions.RemoteControl;

public class BRCMIDlet extends MIDlet {
    private Display currentDisplay;
    private MyRemoteControl control;
    private MyCanvas canvas;
    private String eventName = "CONTROL LOST";
    private int count;
    public BRCMIDlet() {
      currentDisplay = Display.getDisplay(this);

      control = new MyRemoteControl();
      canvas = new MyCanvas();
      currentDisplay.setCurrent(canvas);
    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
    }

    protected void pauseApp() {

    }

    protected void startApp() throws MIDletStateChangeException {

    }


    private class MyCanvas extends Canvas{

      protected void paint(Graphics g) {
        if(control.hasControl()){
          g.setColor(230,255,230);
        }else{
```

```
      g.setColor(255,230,230);
    }

    g.fillRect(0, 0, this.getWidth(), this.getHeight());
    g.setColor(0x000000);
    g.drawString("Has Control: " + control.hasControl(), 10, 10, 0);
    g.drawString("-----------------------------", 10, 20, 0);
    g.drawString("Event: " + eventName, 10, 30, 0);

  }

  protected void keyReleased(int keyCode) {
    if(keyCode == 50){
      control.gainControl();
    }else
    if(keyCode == 49){
      control.loseControl();
    }
  }
}


private class MyRemoteControl extends RemoteControl{
  protected void handleEvent(int event) {
    switch (event) {
    case RemoteControl.BACKWARD_PRESS:
      count++;
      eventName = "BACKWARD PRESS [" + count + "]";
      break;

    case RemoteControl.BACKWARD_RELEASE:
      count = 0;
      eventName = "BACKWARD RELEASE";
      break;

    case RemoteControl.CONTROL_GAINED:
      eventName = "CONTROL GAINED";
      break;

    case RemoteControl.CONTROL_LOST:
      eventName = "CONTROL LOST";
      break;

    case RemoteControl.EXIT_PRESS:
      count++;
      eventName = "EXIT PRESS  [" + count + "]";
      break;

    case RemoteControl.EXIT_RELEASE:
      count = 0;
      eventName = "EXIT RELEASE";
      break;

    case RemoteControl.FAST_FORWARD_PRESS:
      count++;
      eventName = "FAST FORWARD PRESS  [" + count + "]";
      break;

    case RemoteControl.FAST_FORWARD_RELEASE:
      count = 0;
      eventName = "FAST FORWARD RELEASE";
      break;

    case RemoteControl.FORWARD_PRESS:
      count++;
      eventName = "FORWARD PRESS  [" + count + "]";
```

```
  break;

case RemoteControl.FORWARD_RELEASE:
  count = 0;
  eventName = "FORWARD RELEASE";
  break;

case RemoteControl.MUTE_PRESS:
  count++;
  eventName = "MUTE PRESS  [" + count + "]";
  break;

case RemoteControl.MUTE_RELEASE:
  count = 0;
  eventName = "MUTE RELEASE";
  break;

case RemoteControl.PAUSE_PRESS:
  count++;
  eventName = "PAUSE PRESS  [" + count + "]";
  break;

case RemoteControl.PAUSE_RELEASE:
  count = 0;
  eventName = "PAUSE RELEASE";
  break;

case RemoteControl.PLAY_PRESS:
  count++;
  eventName = "PLAY PRESS  [" + count + "]";
  break;

case RemoteControl.PLAY_RELEASE:
  count = 0;
  eventName = "PLAY RELEASE";
  break;

case RemoteControl.REWIND_PRESS:
  count++;
  eventName = "REWIND PRESS  [" + count + "]";
  break;

case RemoteControl.REWIND_RELEASE:
  count = 0;
  eventName = "REWIND RELEASE";
  break;

case RemoteControl.STOP_PRESS:
  count++;
  eventName = "STOP PRESS  [" + count + "]";
  break;
case RemoteControl.STOP_RELEASE:
  count = 0;
  eventName = "STOP RELEASE";
  break;

case RemoteControl.VOLUME_DOWN_PRESS:
  count++;
  eventName = "VOLUME DOWN PRESS  [" + count + "]";
  break;

case RemoteControl.VOLUME_DOWN_RELEASE:
  count = 0;
  eventName = "VOLUME DOWN RELEASE";
  break;

case RemoteControl.VOLUME_UP_PRESS:
```

```
            count++;
            eventName = "VOLUME UP PRESS  [" + count + "]";
            break;

          case RemoteControl.VOLUME_UP_RELEASE:
            count = 0;
            eventName = "VOLUME UP RELEASE";
            break;

          default:
            break;
          }
          canvas.repaint();
        }
      }
}
```

Appendix A: System Properties

Java.lang implementation

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

The additional system properties are:

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

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

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

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

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

Code Sample 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 5: Java ME system properties

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

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

Motorola getsystemProperty() keys for Motorola devices

The table that follows contains the Motorola getsystemProperty() keys. However, not all properties are available on all Motorola 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 6: Motorola getsystemProperty() keys

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 6: Motorola getsystemProperty() keys (Continued)

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

Table 6: Motorola getsystemProperty() keys (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	

Appendix B: Key Mapping

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

Table 25: 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	Soft1 (Menu)
Send	Select (Also, a call is placed if lcdui.TextField or lcdui.TestBox is pressed with PHONENUMBER constraint set). See the note that follows this table for more information.
Center Select	Select

Table 25: Key Mapping

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

NOTE: The MOTOMING A1200 handset uses the Select key to invoke PHONENUMBER select dialog in TextBox. For keypad handsets, the select key is used for other purposes and the Send key is used to invoke PHONENUMBER select dialog.

Appendix C: 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 26 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 26: MIDlet Attributes, Descriptions, JAD, and JAR Manifest

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlet to the user.	Yes	Yes
MIDlet-Version	The version number of the MIDlet suite.	Yes	Yes
MIDlet-Vendor	The organization that provides the MIDlet suite.	Yes	Yes
MIDlet-Icon	The case-sensitive absolute name of a PNG file within the JAR, used to represent the MIDlet suite.	Yes	Yes
MIDlet-Description	The description of the MIDlet suite.	No	No
MIDlet-Info-URL	A URL for further information describing the MIDlet suite.	Yes	No
MIDlet-<n>	The name, icon, and class of the nth MIDlet in the JAR file. The name identifies this MIDlet to the user. Icon is as stated above. Class is the name of the class extending the `javax.microedition.midlet.MIDletclass`.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR manifest.
MIDlet-Jar-URL	The URL from which the JAR file is loaded.		Yes
MIDlet-Jar-Size	The number of bytes in the JAR file.		Yes

Table 26: MIDlet Attributes, Descriptions, JAD, and JAR Manifest (Continued)

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Data-Size	The minimum number of bytes of persistent data required by the MIDlet.	Yes	Yes
MicroEdition-Profile	The Java™ ME profiles required. If any of the profiles are not implemented, the installation fails.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR manifest.
MicroEdition-Configuration	The Java™ ME Configuration required, that is, CLDC.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR manifest.
MIDlet-Permissions	Zero or more permissions that are critical to the function of the MIDlet suite.	Yes	Yes
MIDlet-Permissions-Opt	Zero or more permissions that are non-critical to the function of the MIDlet suite.	Yes	Yes
MIDlet-Push-<n>	Register a MIDlet to handle inbound connections.	Yes	Yes
MIDlet-Install-Notify	The URL to which a POST request is sent to report installation status of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Notify	The URL to which a POST request is sent to report deletion of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Confirm	A text message to be provided to the user when prompted, to confirm deletion of the MIDlet suite.	Yes	Yes

Appendix D: 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

Action	Resulting action
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'.
Failed file Corrupt	During Installation, if the MANIFEST file is wrong, the handset displays a transient notice 'Failed File Corrupt'.
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.
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 27 shows the error logs associated with downloading MIDlets.

Table 27: 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
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

Table 27: Error logs (Continued)

Error Dialog	Scenario	Possible Cause	Install-Notify
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

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.
Download Cancelled	A downloading application can be cancelled by pressing the END key. The transient notice, 'Download Cancelled,' is displayed. 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.
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.”