Technical Manual

MOTOROLA

Developer Guide

MOTOROKR E6/E6e Java™ ME Developer Guide

Version 02.00

Copyright © 2007, Motorola, Inc. All rights reserved. This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications, can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application. No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected. In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore the above limitations or exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur. Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, the buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacturing of the product or service. Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

MOTOROLA and the Stylized M Logo are registered in the U.S. Patent & Trademark Office. All other product or service names are the property of their respective owners. Java and all other Java based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc.

MOTOROKR E6/E6e Java™ ME Developer Guide
Version 02.00
11-APR-2007
For the latest version of this document, visit *http://developer.motorola.com*.
Motorola, Inc.
http://www.motorola.com

Table of Contents

1 Java™ ME Introduction

The MOTOROKR E6/E6e handset includes the Java Micro Edition (Java™ ME) Platform, Micro Edition, also known as the Java™ ME platform. The Java™ ME platform enables developers to easily create a variety of Java applications ranging from business applications to games. Prior to its inclusion, services or applications residing on small consumer devices like cell phones could not be upgraded or added to without significant effort. By implementing the Java™ ME platform on devices like the MOTOROKR E6/E6e handset, service providers, as well as customers, can easily add and remove applications allowing for quick and easy personalization of each device. This chapter presents a quick overview of the Java™ ME environment and the tools that can be used to develop applications for the MOTOROKR E6/E6e handset.

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

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

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

typically encompassing the presentation layer of the Java Platform.

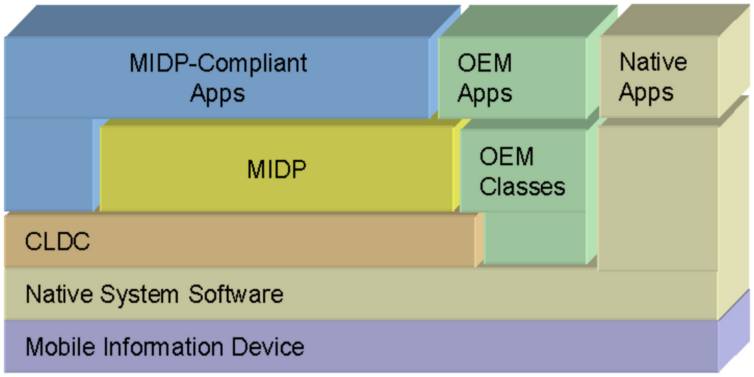


Figure 1 Layer Architecture

The Configuration Layer used in the iDEN handsets is either the Connected Limited Device Configuration 1.1 (CLDC 1.1) or the Connected Limited Device Configuration 1.0 (CLDC 1.0), depending on the phone model. The Profile Layer used is the Mobile Information Device Profile 2.0 (MIDP 2.0). Together, the CLDC and MIDP provide common APIs for I/O, simple math functionality, UI, and more.

For more information on Java™ ME see the Sun™ Java™ ME documentation (*http://java.sun.com/javame*).

The Motorola Java™ ME Platform

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

The MOTOROKR E6/E6e handset contains OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the MOTOROKR E6/E6e handset can run any application written in standard MIDP, it can also run applications that take advantage of the unique functionality provided by these APIs. These OEM APIs are described in this guide.

Resources and APIs Available

MIDP 2.0 provides support to the following functional areas on the MOTOROKR E6/E6e handset:

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

Additional Functionality:

- JSR-118
- JSR-120
- JSR-135
- JSR-139
- JSR-172
- JSR-184
- JSR-185
- JSR-205
- JSR-75 FileConnection API
- JSR-75 PIM API
- JSR-82
- Motorola Get URL from Flex API

2 Developing and Packaging Java™ ME Applications

Guide to Development in Java™ ME

Introduction to Development

This chapter assumes you have previous experience in Java™ ME development and can appreciate the development process for Java MIDlets. This chapter provides some information that a beginner in development can use to gain an understanding of MIDlets for Java™ ME handsets.

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

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

As an introduction, brief details of Java™ ME are explained below.

The MIDlet consists of two core specifications, namely Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). Both of these specifications (JSR - Java Specification Requests) are located at *http://www.jcp.org/* site for reading.

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

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

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

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

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

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

3 Downloading Applications

Methods of Downloading

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

Method 1 - OTA

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

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

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

- HTTP Range - see specification RFC 2068 at *http://www.rfc-editor.org/rfc.html* if content greater than 30k in size. Below is a ladder diagram showing the flow

through HTTP range transfer, although recall use of the .JAD is optional.
- SAR (Segmentation & Reassembly) - see specification of wireless transaction protocol at the *http://www.wapforum.org* if less than 30k in size.

During a download of the application, the user will see the Opera 8 displaying 'Downloading' followed by "x% completed" for either SAR or HTTP Byte Range type downloads.

A basic Over the Air Server Configuration document can be found in our Technical Articles section at *http://developer.motorola.com*. This includes details of configuring the server and also example WAP pages.

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

The following error codes are supported:

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

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

Possible Screen Messages Seen With Downloading:

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

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

The USER-AGENT String

Table 1 describes USER_AGENT strings associated with Motorola devices:

Motorola Device	USER_AGENT STRING
MOTOROKR E6/E6e	MOT-E6/xx.xx.xxR Opera/8 Profile/MIDP-2.0 Configuration/CLDC-1.1

Table 1 USER_AGENT String

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

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

Error Logs

Table 2 represents the error logs associated with downloading applications.

Error Dialog	Scenario	Possible Cause	Install-Notify
Failed: Invalid File	JAD Download	Missing or incorrectly formatted mandatory JAD attributes Mandatory: MIDlet-Name MIDlet-Version MIDlet-Vendor MIDlet-JAR-URL 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 valid-	903 Loss of Service

		ated JAD signature verification failed Unknown error during JAD validation See 'Details' field in the dialog for information about specific error	
Insufficient Storage	OTA JAR Download	Insufficient data space to temporarily store the JAR file	901 Insufficient Memory
Application Already Exists	OTA JAR Download	MIDlet version numbers are identical	905 Attribute Mismatch
Different Version Exists	OTA JAR Download	MIDlet version on handset supercedes version being downloaded	
Failed File Corrupt	Installation	Attributes are not identical to respective JAD attributes	
Insufficient Storage	Installation	Insufficient Program Space or Data Space to install suite	901 Insufficient Memory
Application Error	Installation	Class references non-existent class or method Security Certificate verification failure Checksum of JAR file is not equal to Checksum in MIDlet-JAR-SHA attribute Application not authorized	
Application Expired	MIDlet Launching	Security Certificates expired or removed	
Application Error	MIDlet Execution	Authorization failure during MIDlet execution Incorrect MIDlet	

Table 2 Error Logs

4 Application Management

The following sections describe the application management scheme for the MOTOROKR E6/E6e handset. This chapter discusses the following:

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

Downloading a JAR File Without a JAD

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

MIDlet Upgrade

Rules from the JSR-118 (MIDP 2.0) are followed to help determine if the data from an old MIDlet should be preserved during a MIDlet upgrade. When these rules cannot determine if the RMS should be preserved, the user is given an option to preserve the data.

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

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

Installation and Deletion Status Reports

The status (success or failure) of an installation, upgrade, or deletion of a MIDlet suite 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's request. Upon successful deletion, the handset sends the status code 912 to the MIDlet-Delete-Notify URL. If this notification fails, the MIDlet suite is still deleted. If this notification cannot be sent due to lack of network connectivity, the notification is sent at the next available network connection.

Table 3 lists the application management feature/class support for MIDP 2.0:

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

Table 3 Application Management Feature

5
iTAP

Intelligent Keypad Text Entry API

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

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

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

The following Java™ ME text input components support iTAP.

- javax.microedition.lcdui.TextBox

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

- javax.microedition.lcdui.TextField

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

Refer to Table 4 for iTAP feature/class support for MIDP 2.0:

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

Table 4 iTAP Feature/Class

6 Record Management System

Record Management System API

If the MIDlet has not specified a data space requirement in the JAD attribute (MIDlet_data_space_requirement) or the manifest file, a limit of 1Mb is used as the maximum RMS space for that MIDlet. No additional Motorola implementation clarifications are necessary.

Table 5 lists the RMS feature/class support for MIDP 2.0:

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

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

Table 5 RMS Feature/Class

7 Downloading MIDlet Through Browser

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

The example shows how to access the Browser application by any of the following methods:

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

Start Active Browser Session from Main Menu

Figure 2 describes Starting Active Browser Session from Main Menu:

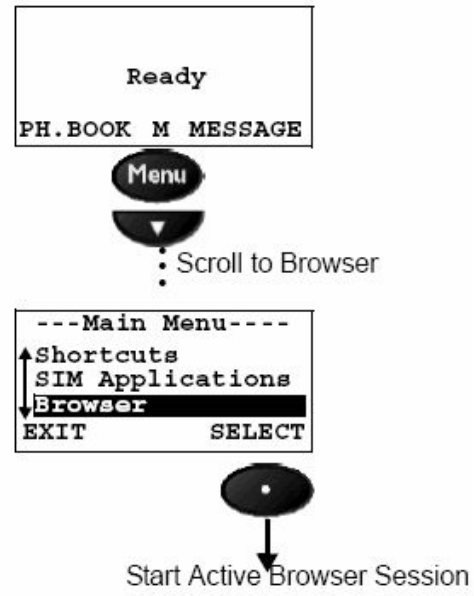


Figure 2 Starting Active Browser Session from Main Menu

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

Find a Location with Java™ ME Application

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

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

Downloading MIDlets

Figure 3 represents Java™ ME Application (MIDlets) Download and Installation.

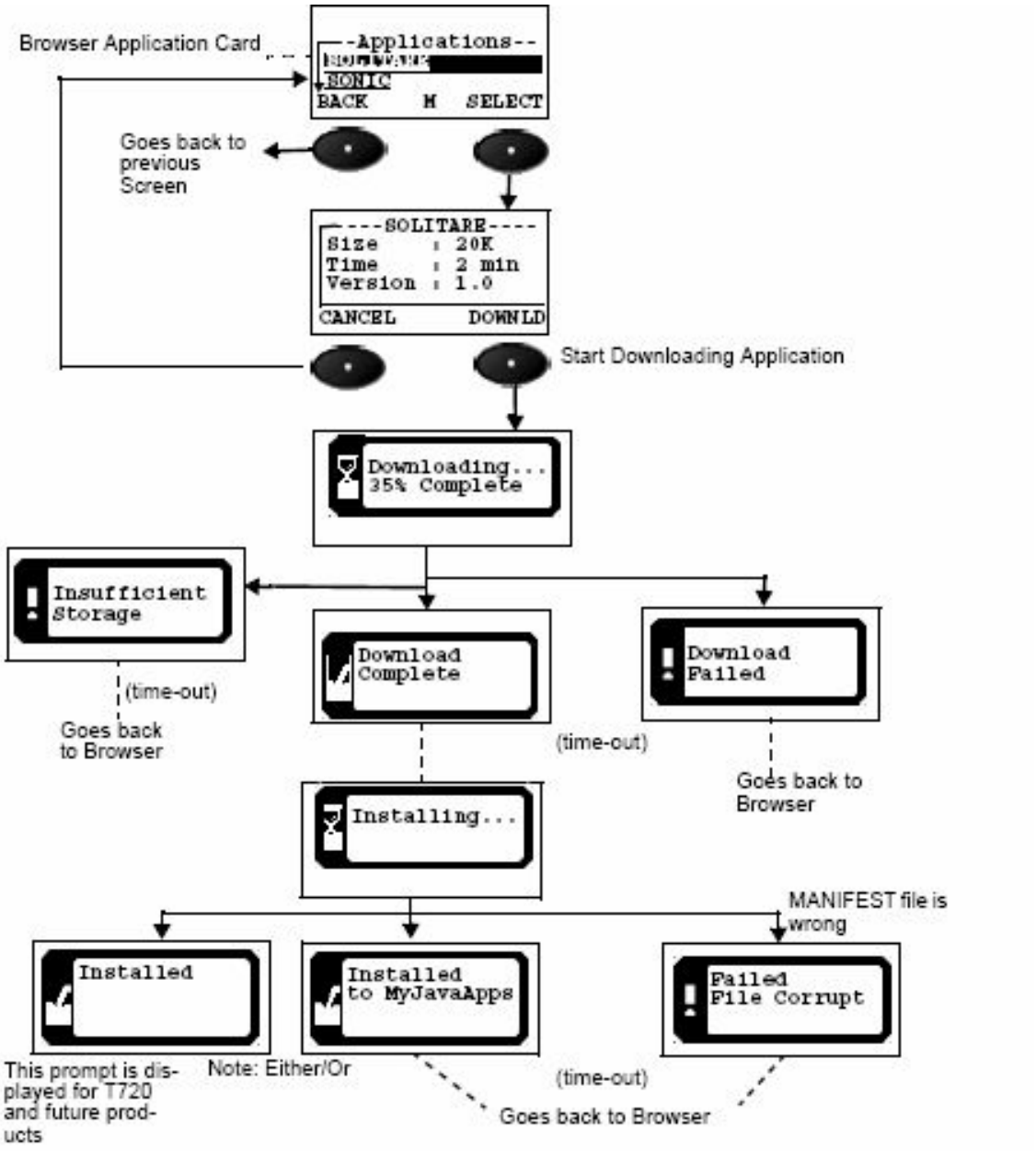


Figure 3 Downloading and Installing Java™ ME Application (MIDlets)

Steps to Download and Install Java™ ME Application:

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

- If the browser connection is interrupted/ended during the download/installation process, the device is unable to send the HTTP POST with the MIDlet-Install Notify attribute. In this case, the MIDlet is deleted to ensure the user does not get a free MIDlet. The use case can occur when a phone call is accepted and terminated during the installation process, because then the browser is not in the needed state to return the MIDlet Install Notify attribute.
- Upon completing Installation, the handset displays a transient notice 'Installed to Games and Apps'.
- Upon time-out, the handset goes back to Browser.
- During Installation if the MANIFEST file is wrong, the handset displays a transient notice 'Failed File Corrupt'. Upon time-out, the handset goes back to Browser.
- During the installation process, if the flip is closed on a flip handset, the installation process 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.
- During download or install of application, voice record, voice commands, voice shortcuts, and volume control is not supported. However, during this time, incoming calls and SMS messages are able to be received.
- The handset must support sending and receiving at least 30 kilobytes of data using HTTP either from the server to the client or the client to the server, per Over the Air User Initiated Provisioning specification.
- If JAD does not contain mandatory attributes, 'Failed Invalid File' notice appears.

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

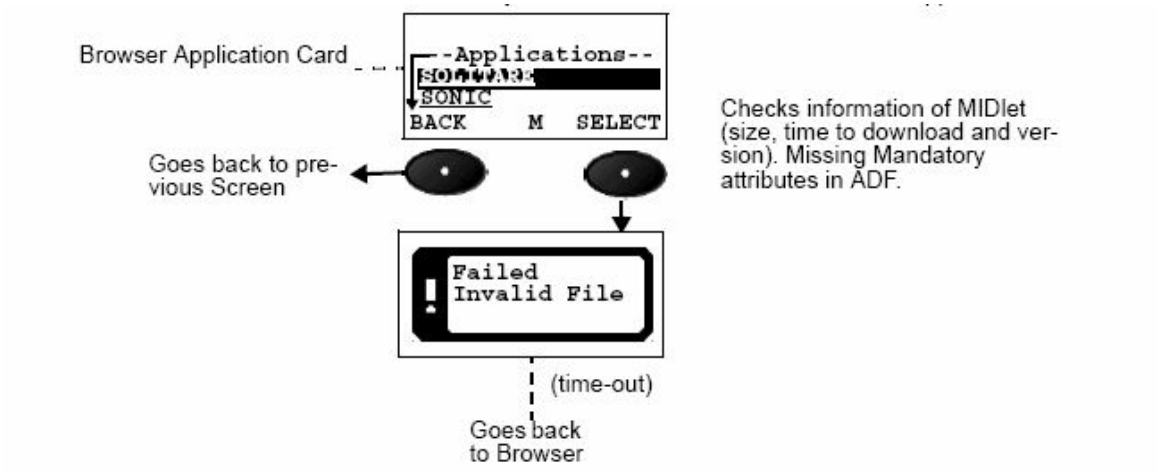


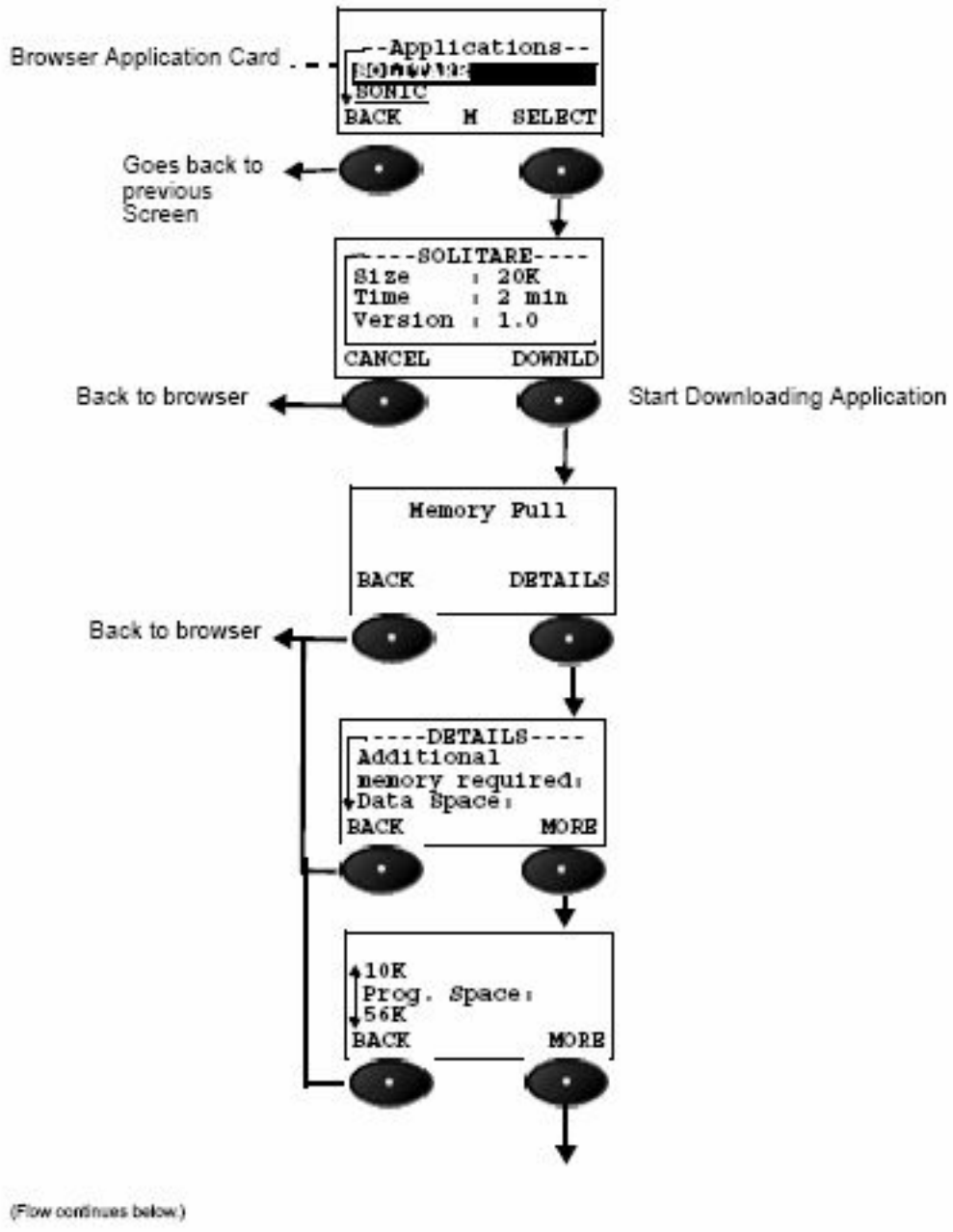
Figure 4 Application Does Not Have Mandatory Attributes in ADF

Different Error Checks

Memory Full

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


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

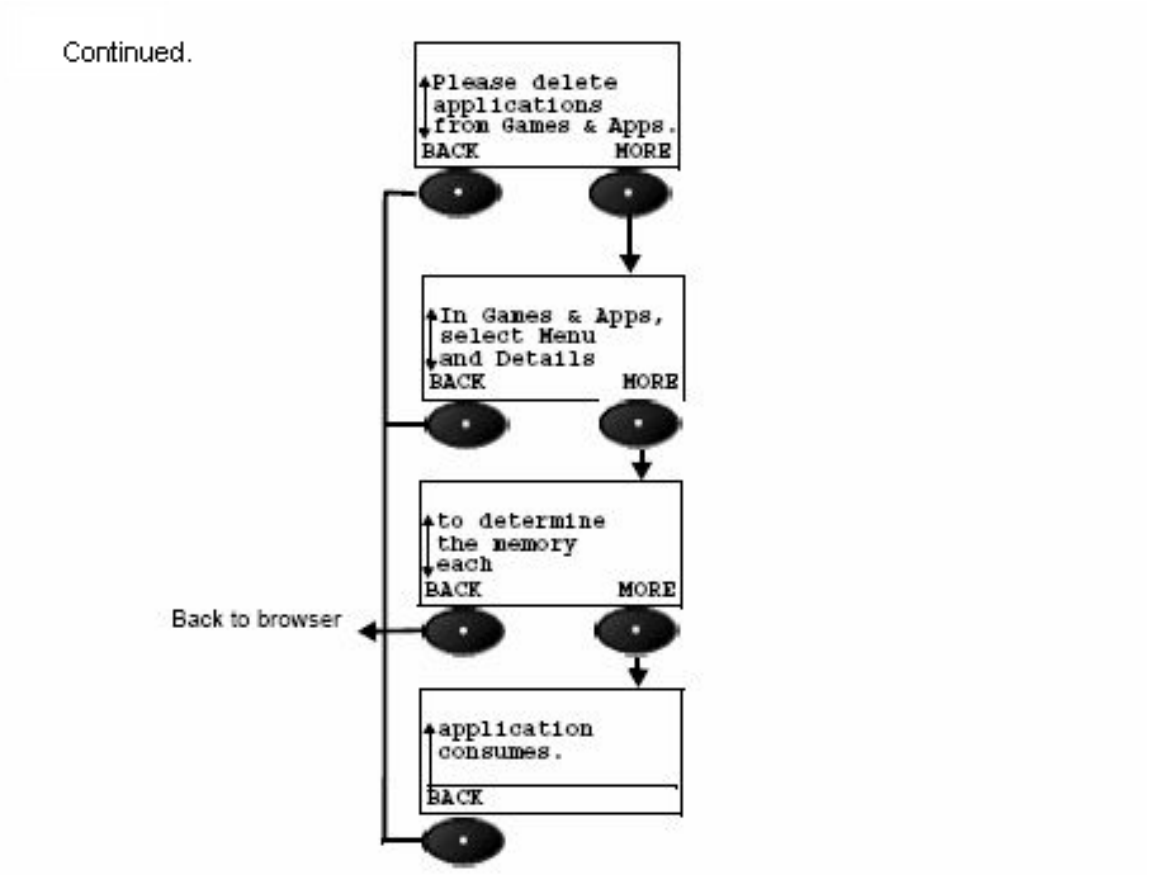


Figure 5 Memory Full Error

Rules:

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

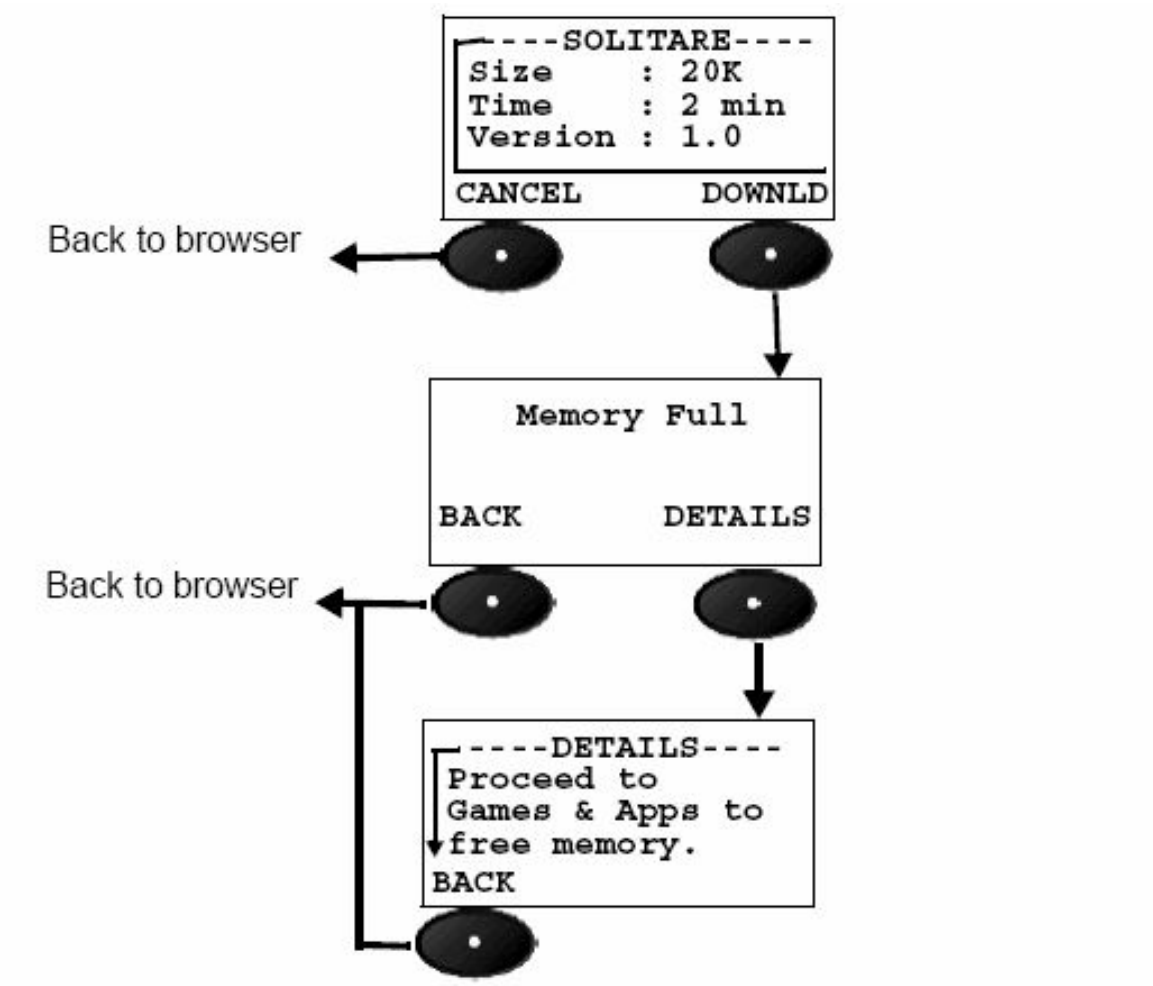


Figure 6 Mot-Data-Space and Mot-Program-Space Attributes

- If Mot-Data-Space-Requirements and Mot-Program-Space-Requirements JAD attributes are not present in the JAD file, the handset can not determine how much memory to free and displays the above help dialog.
- 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 should disappear when the user has scrolled to the bottom of the dialog.
- All rules stated in the previous figure must also be followed for the above stated prompt.

Memory Full During Installation Process

Once the MIDlet is successfully downloaded, the installation process begins. During the installation of the MIDlet, the phone may determine there is insufficient memory to complete the installation. This error can occur whether the Mot-Data-Space-Requirements and Mot-Program-Space-Requirements JAD attributes are present or not. The following message and Figure Figure 7 must be displayed:

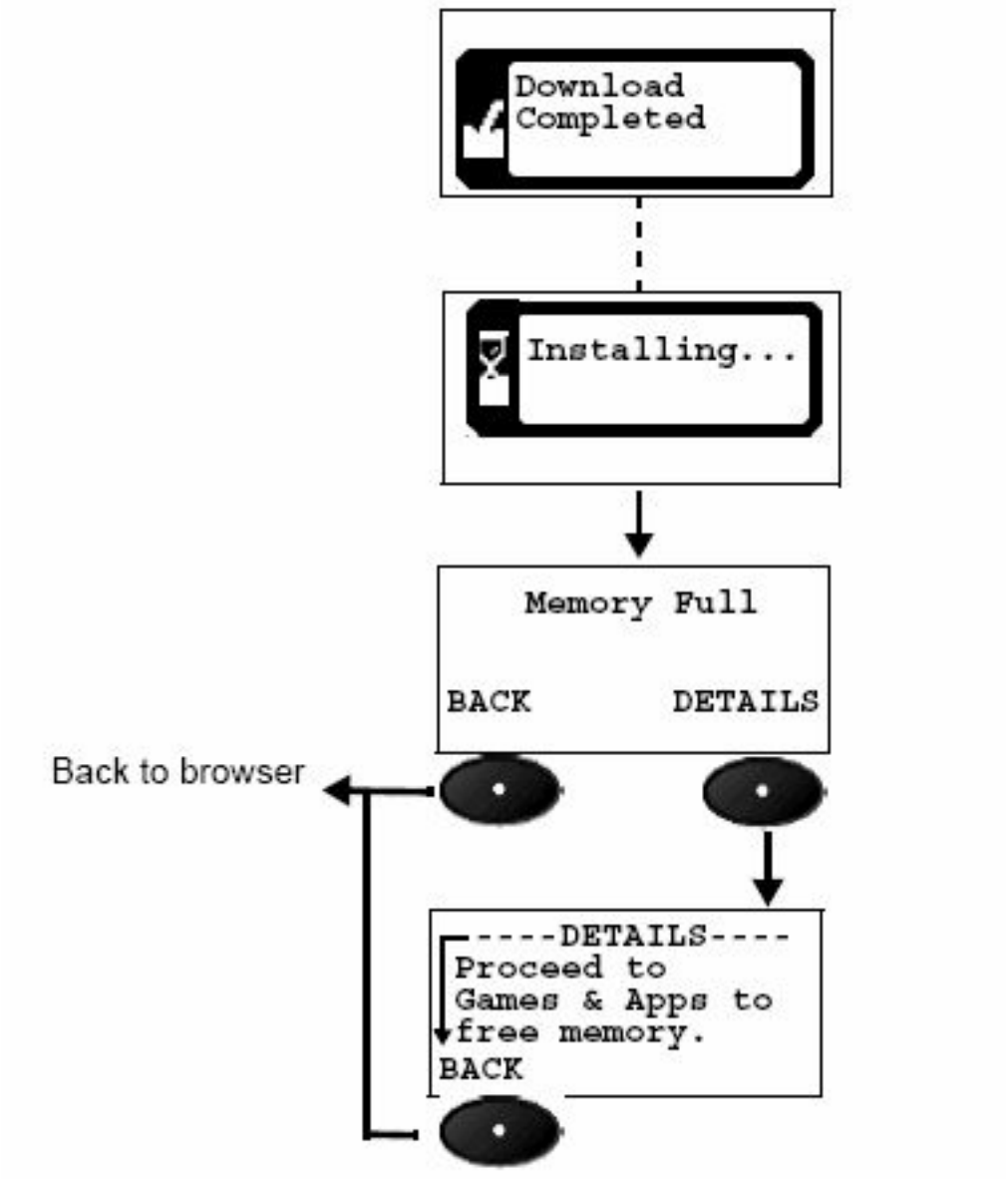


Figure 7 Memory Full Help Message During Installation Process

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

Application Version Already Exists

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

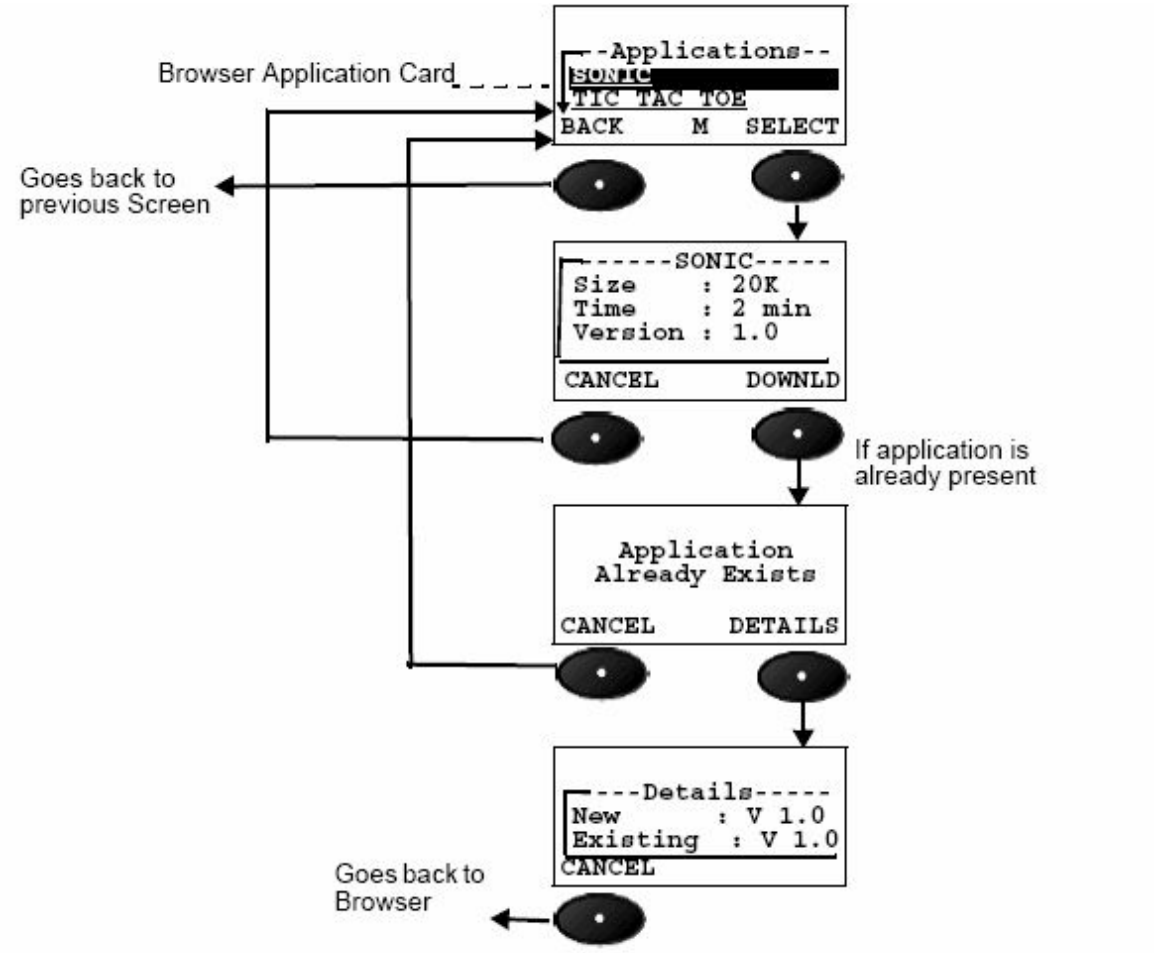


Figure 8 Same Version of Application Already Exists on the Handset

Rules:

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

Newer application version exists

If the application version on the handset is newer than the downloaded version of application, the following message is displayed. The error occurred can be queried by selecting DETAILS.

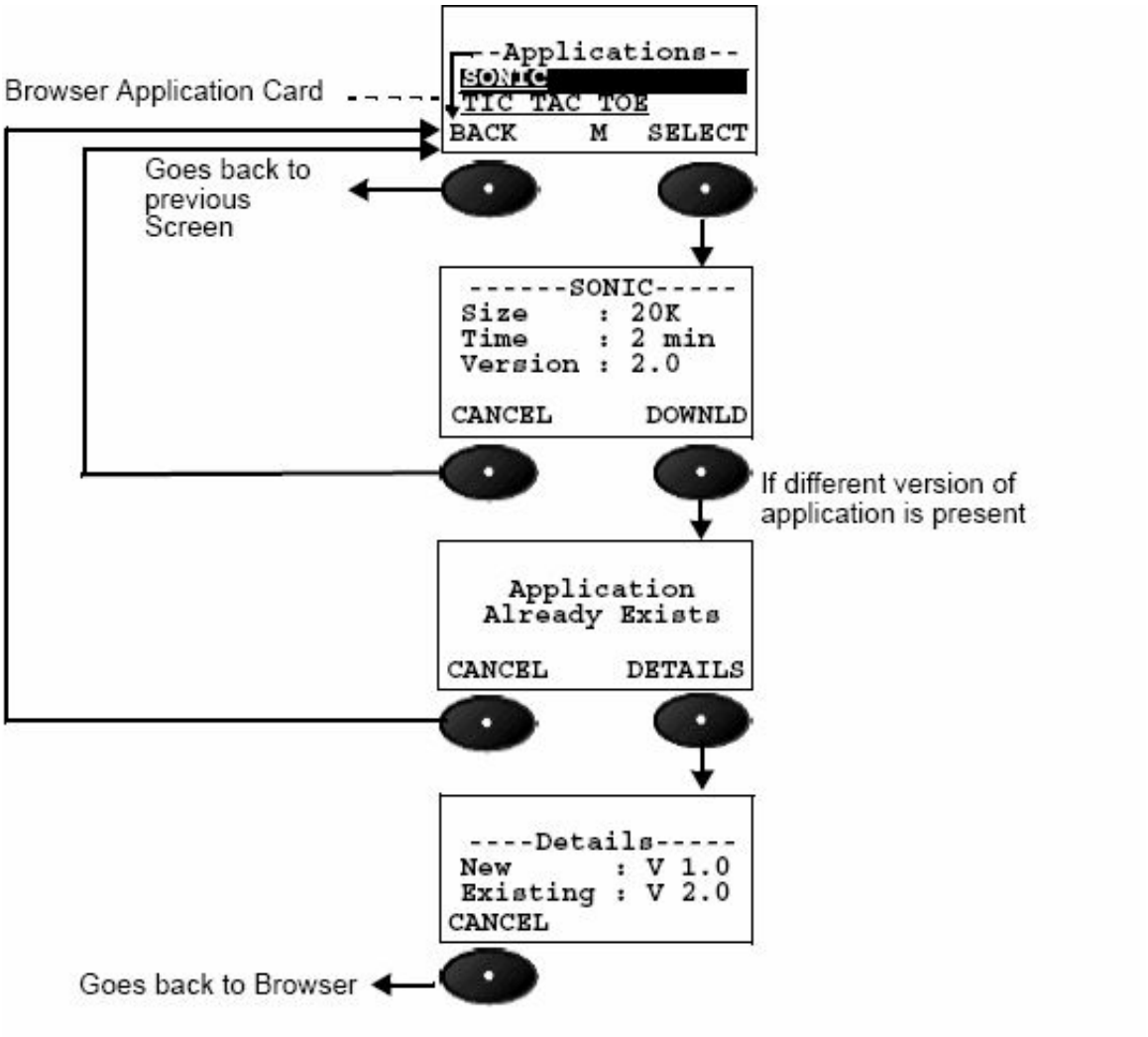


Figure 9 (Newer) Version of Application Exists

Rules:

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

8
JAD Attributes

JAD / Manifest Attribute Implementations

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

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

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

Motorola has implemented the following support for the MIDP 2.0 Java Application Descriptor attributes as outlined in the JSR-118. Table 6 lists all MIDlet attributes, descriptions, and its location in the JAD and/or JAR manifest that are supported in the Motorola implementation. Please note that the MIDlet does not install if the MIDlet-Data-Size is greater than 512k.

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlets to the user.	Yes	Yes
MIDlet-Version	The version number of the MIDlet suite.	Yes	Yes
MIDlet-Vendor	The organization that provides the MIDlet suite.	Yes	Yes
MIDlet-Icon	The case-sensitive absolute name of a PNG file within the JAR, used to represent the MIDlet suite.	Yes	Yes
MIDlet-Description	The description of the MIDlet suite.	No	No
MIDlet-Info-URL	A URL for information further describing the MIDlet suite.	Yes	No
MIDlet-<n>	The name, icon, and class of the *n*th MIDlet in the JAR file. Name is used to identify this MIDlet to the user. Icon is as stated above. Class is the name of the class extending the `javax.microedition.midlet.MIDletclass`.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR manifest.
MIDlet-Jar-URL	The URL from which the JAR file is loaded.		Yes
MIDlet-Jar-Size	The number of bytes in the JAR file.		Yes
MIDlet-Data-Size	The minimum number of bytes of persistent data required by the MIDlet.	Yes	Yes
MicroEdition-Profile	The Java™ ME profiles required. If any of the profiles are not implemented the installation 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 in-bound connections	Yes	Yes
MIDlet-Install-Notify	The URL to which a POST request is sent to report installation status of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Notify	The URL to which a POST request is sent to report deletion of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Confirm	A text message to be provided to the user when prompted to confirm deletion of the MIDlet suite.	Yes	Yes

Table 6 MIDlet Attributes, Descriptions, and its Location in the JAD and/or JAR Manifest

9 Network APIs

Network Connections

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

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

Refer to Table 7 for Network API feature/class support for MIDP 2.0:

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

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

Table 7 Network API feature/class support for MIDP

Code Sample 1 shows the implementation of Socket Connection:

Socket Connection

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

...

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

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

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

          //read data from server until done
```

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

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

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

Code Sample 1 Socket Connection

User Permission

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

Indicating a Connection to the User

When the java implementation makes any of the additional network connections, it will indicate to the user that the handset is actively interacting with the network. To indicate this connection, the network icon will appear on the handset's status bar as shown in Figure 10 .

Figure 10 Network Connections example

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

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

HTTPS Connection

Motorola implementation supports a HTTPS connection on the MOTOROKR E6/E6e handset. Additional protocols that will be supported are the following:

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

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

Code Sample 2 shows the implementation of HTTPS:

HTTPS

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



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

      } catch (Exception ex) {
```

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

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

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

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

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

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

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

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

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

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

Code Sample 2 HTTPS

DNS IP

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

Push Registry

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

Mechanisms for Push

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

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

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

Push Registry Declaration

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

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

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

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

Dynamically register using an assigned port number

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

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

Code Sample 3 shows the implementation of Push Registry:

Push Registry Declaration

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

public class PushTest_1 extends MIDlet implements CommandListener{

        public Display display;
```

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

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

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

    public static Command mc;
    public static Displayable ms;

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

        display = Display.getDisplay(this);

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

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

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

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

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

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

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


    }
    public void pauseApp(){}


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

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

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

    public void commandAction(Command c, Displayable s) {

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

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

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

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

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

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

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

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

      }

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

            }
```

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

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

WakeUp.java

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

```
public class WakeUp extends MIDlet implements CommandListener{

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


    public void startApp() {

        display = Display.getDisplay(this);

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



        display.setCurrent(mainForm);

    }

    public void pauseApp() {
    }

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

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

            new regThread().start();

        }
    }
```

```
   public class regThread extends Thread{


      public void run(){

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

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

            System.out.println(curTime + delay);

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

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

      }
   }
}
```

SMS_send.java

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

public class SMS_send extends MIDlet implements CommandListener{

      public Display display;

      public static Form messageForm;
      public static Form mainForm;

      public static Command exitCommand;
```

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

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

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

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

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

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

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


            display = Display.getDisplay(this);

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

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

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

            messageForm.addCommand(backCommand);
```

```
        messageForm.setCommandListener(this);


    }

    public void pauseApp(){
    }

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

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

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

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

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

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

Code Sample 3 Push Registry

Delivery of a Push Message

A push message intended for a MIDlet on the MOTOROKR E6/E6e handset will handle the following interactions:

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

No MIDlet suites running - if no MIDlets are running, the user will be notified of the incoming push message and will be given the option to run the intended application as shown in Figure 11 .

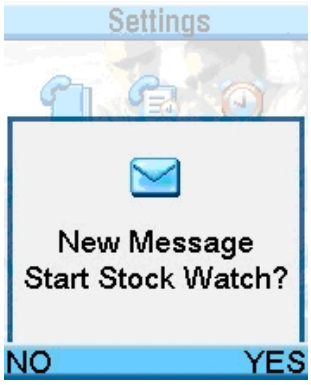


Figure 11 Intend Application Run Option

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

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

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

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

Deleting an Application Registered for Push

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

Security for Push Registry

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

Network Access

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

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

10 Platform Request API

Platform Request API

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

Table 8 lists the Platform Request API feature/class support for MIDP 2.0:

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

Table 8 Platform Request API Feature/Class Support for MIDP

For MIDP 2.0, the javax.microedition.midlet.MIDlet.platformRequest() method is used and called when the MIDlet is destroyed. The following code sample is an example of the Platform Request API:

Start a Call

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

Start a Web Session

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

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

Code Sample 4 Plataform Request

MIDlet Request of a URL That Interacts with Browser

When a MIDlet suite requests a URL, the browser comes to the foreground and connects to that URL. The user has access to the browser and control over any downloads or network connections. The initiating MIDlet suite continues running in the background. If it cannot (upon exiting the requesting MIDlet suite) the handset brings the browser to the foreground with the specified URL.

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

Refer to the *JAD Attributes* for more details.

MIDlet Request of a URL That Initiates a Voice Call

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

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

11
JSR-75: PIM API

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

NOTE: Java™ ME PIM API is implemented on Java ME platforms supporting CLDC 1.1 and MIDP 2.0 or higher.

Overview

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

This chapter details requirements for implementing the PIM API specified in JSR-75 for Java ME enabled mobile devices.

This implementation provides the basic features available as part of the standard JSR 75 PIM implementation. It is available as the `javax.microedition.pim package`.

Requirements

The implementation includes support of the following packages, classes, and interfaces with appropriate methods and fields of PIM API described in JSR-75, related to `javax.microedition.pim`:

- javax.microedition.pim.PIM;
- javax.microedition.pim.RepeatRule;
- javax.microedition.pim.PIMException;
- javax.microedition.pim.FieldEmptyException;
- javax.microedition.pim.FieldFullException;
- javax.microedition.pim.UnsupportedFieldException;
- javax.microedition.pim.PIMItem;
- javax.microedition.pim.Contact;
- javax.microedition.pim.Event;
- javax.microedition.pim.PIMList;
- javax.microedition.pim.ContactList;
- javax.microedition.pim.EventList;

The implementation includes support of the following packages, classes, and interfaces with appropriate methods and fields of FileConnection API described in JSR-75, related to `javax.microedition.io.file`:

- javax.microedition.io.file.ConnectionClosedException;
- javax.microedition.io.file.IllegalModeException;
- javax.microedition.io.file.FileConnection.

Security Requirements Personal information read/write permissions are supported by the device's native system:

- `javax.microedition.pim.ContactList.read` — enables reading the contact information available on the device (hereinafter just "contact read permission").
- `javax.microedition.pim.ContactList.write` — enables updating the contact information available on the device (hereinafter just "contact write permission").
- `javax.microedition.pim.EventList.read` — enables reading the event information available on the device (hereinafter just "event read permission").
- `javax.microedition.pim.EventList.write` — enables updating the event information available on the device (hereinafter just "event write permission").

The PIM permissions are mapped to the function groups "User Data Read Capability" and "User Data Write Capability," depending on the read/write conditions. These two groups and the permissions are in Table 9:

Function	Trusted Third	Untrusted	Manufacturer	Operator

Group	Party			
User Data Read Capability	Always Ask, Ask Once Per App, Never Ask, No Access	Always Ask	Full Access	Full Access
User Data Write Capability	Always Ask, Ask Once Per App, Never Ask, No Access	No Access	Full Access	Full Access

Table 9 Permissions and Groups

The PIM permissions prohibit granting to a MIDlet suite that does not request them explicitly in the attributes MIDlet-Permissions or MIDlet-Permissions-Opt.

The PIM package allows the handling of two types of lists: events and contacts lists. Both are stored in a specific database respectively: event database and contact database. These databases have specific information of each list.

Table 10 Features x JSR-75 PIM

Feature	JSR 75 PIM
Name Addition to a Contact	Yes
Phone Number Addition to a Contact	Yes
Email Addition to a Contact	Yes (1)
Multiple Email and Phone number Addition	No
ToDo List	No

Table 10 Features x JSR-75 PIM

NOTE: Uses `com.motorola.pim.Contact.MOT_EMAIL`.

Fields and Attributes

Contact List

The contact database contains data items representing personal contact information (like name, address, etc). The following features are applied to the contact list:

- The implementation provides support for ContactList type of PIM list as defined in JSR-75.
- The implementation provides a method to access an actual list of the PIM ContactList type.
- The implementation provides interface to manipulate actual ContactList as specified in ContactList class section of JSR-75.
- The implementation provides access to all available PIM lists for the ContactList list type.
- At a minimum, the following fields are supported : ADDR, BIRTHDAY, FORMATTED_NAME, NICKNAME, TELEPHONE, MOT_EMAIL, UID, CONTACT_TYPE, MOT_PHOTO_URL, LIGHT_ID, LOCATION, MEMBER_IDS, VOICE TAG, and RINGTONE.
- At a minimum, the following attributes are supported: ATTR_PAGER, ATTR_MOBILE, ATTR_OTHER, ATTR_HOME, ATTR_WORK, ATTR_FAX, ATTR_VIDEO, and ATTR_NONE.
- The location of the contact information (that is, SIM card or Phone Memory) is defined by separate dedicated field content value.

Table 11 Supported Fields for the `Contact`.

Field Description	JSR 75 PIM Field
Contact Address	ADDR
Birthdate	BIRTHDAY
Contact Name	FORMATTED_NAME
Contact Nickname	NICKNAME
Contact Telephone Number	TELEPHONE
Contact Email	MOT_EMAIL (Motorola Extended)
Contact Unique ID	UID
Contact Type (Phone, SIM, Mailing List)	CONTACT_TYPE (Motorola Extended)
Contact Photo URL	MOT_PHOTO_URL (Motorola Extended)
Contact Light ID	LIGHT_ID (Motorola Extended)
Contact Location	LOCATION (Motorola Extended)
Mailing List Member IDs	MEMBER_IDS (Motorola Extended)
Contact Voice Tag	VOICE TAG (Motorola Extended)

Contact Ringtone URL	RINGTONE (Motorola Extended)

Table 11 Contact List - Fields - JSR-75 PIM

Table 12 Supported Attributes for some of fields of `Contact`.

Field Label	Attributes in JSR 75 PIM
TELEPHONE	ATTR_MOBIL, ATTR_WORK, ATTR_HOME, ATTR_FAX, ATTR_PAGER, ATTR_NONE
EMAIL	ATTR_NONE
ADDR	ATTR_NONE

Table 12 Contact List - Attributes - JSR-75 PIM

Event List

The event database contains entries related to events (for example, birthday). The following features are applied to the contact list:

- The implementation provides support for EventList type of PIM list as defined in JSR-75.
- The implementation provides a method to access an actual list of the PIM EventList type.
- The implementation provides an interface to manipulate the actual EventList as specified in the EventList class section of JSR-75.
- The implementation provides access to all available actual PIM lists for the EventList list type.
- At a minimum, the following Event fields are supported: SUMMARY, UID, END, START, and ALARM.
- At a minimum, the following repeat rules fields are supported: FREQUENCY, DAY_IN_WEEK, WEEK_IN_MONTH, and DAY_IN_MONTH.
- At a minimum, one attribute is supported: ATTR_NONE.

Table 13 Fields supported for Event items.

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

Table 13 Event List - JSR-75 PIM

ToDo List

ToDo is only supported by the JSR-75 PIM Enhacement 3G implementation.

12 JSR-75: FileConnection API

Overview

The primary goal of the FileConnection API is to provide access to file systems on devices and/or mounted removable memory cards supported by Motorola devices. This API is not meant to be a replacement for the Record Management System (RMS) but rather a complement to it allowing MIDlets to interact with native applications.

NOTE: Java™ ME FileConnection API is implemented on Java ME platforms supporting CLDC 1.1 and MIDP 2.0 or higher.

Requirements

FileConnection API requirements are replaced with the requirements below.

- The implementation provides a security model for accessing the FileConnection API.
- The FileConnection API is accessible to manufacturer and operator domain MIDlets, subject to security restrictions.
- Connection API prohibits the modification or removal of files and directories marked with the system attribute.
- Call to `System.getProperty()` with key `microedition.io.file.FileConnection.version` returns the implementation version number, starting with 1.0.

Interface

The FileConnection API contains one class, two interfaces, and two exceptions. The most important one of these is the FileConnection interface, which extends the Connection interface. This interface is intended to access files or directories that are located on removable media and/or file systems on a device.

There are two ways to access the file system: through Generic Connection Framework (GCF) or using FileConnection to write/read files.

When GCF is used, the format of the input string used to access a FileConnection through `Connector.open()` must follow the format for a fully qualified, absolute file name. This format has the following structure: "file://<host>/<path>".

Some examples of opening a FileConnection from a root value are in Table 14:

Possible Root Value	Opening a FileConnection to the Root
CFCard/	`Connector.open("file:///CFCard/")`
SDCard/	`Connector.open("file:///SDCard/")`
MemoryStick/	`Connector.open("file:///MemoryStick/")`
C:/	`Connector.open("file:///C:/")`
/	`Connector.open("file:////")`

Table 14 Opening a FileConnection

Security

File operations are restricted with the aim of protecting the user's private data and the overall system security. File operations can be executed only if the required permission has been acquired before. Implementations *must not* allow a FileConnection to access MIDP RMS databases and *should not* allow access to files and configuration files, device and OS specific files and directories. If the file, file system, or directory is not allowed to be accessed, a `java.lang.SecurityException` is thrown from the `Connector.open()` method.

Permissions

Two permissions have been defined in relation to FileConnection API:

- `javax.microedition.io.Connector.file.read` — enables reading from the file system (hereinafter just "read permission").
- `javax.microedition.io.Connector.file.write` — enables writing to the file system (hereinafter just "write permission").

The "read permission" and "write permission" are mapped to the function groups "User Data Read Capability" and "User Data Write Capability," respectively. These two groups and permissions are in Table 15:

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

Table 15 Groups and permissions for

The FileConnection permissions are prohibited for granting to a MIDlet suite that doesn't request them explicitly in the attributes MIDlet-Permissions or MIDlet-Permissions-Opt.

If the permission is not granted, a `SecurityException` is thrown by the following methods: `open`, `openDataInputStream`, `openInputStream`, `setFileConnection`, `listRoots`, `openDataOutputStream`, `openOutputStream`, `PIMList.items(all methods)`, `PIMList.itemsByCategory`, `PIMList.addCategory`, `PIMList.deleteCategory`, `PIMList.renameCategory`, `PIMItem.commit`, `ContactList.removeContact`, `EventList.removeEvent`, and `EventList.items`.

The following `javax.microedition.io.Connector` methods check for the "read permission":

- `open("file:///...")`
- `open("file:///...", Connector.READ)`
- `open("file:///...", Connector.READ_WRITE)`
- `openDataInputStream()`
- `openInputStream()`

The following `javax.microedition.io.file.FileConnection` methods check for the "read permission":

- `setFileConnection`, when instance opened with READ;
- `setFileConnection`, when instance opened with READ_WRITE.

The following `javax.microedition.io.Connector` methods check for the "write permission":

- `open("file:///...")`
- `open("file:///...", Connector.WRITE)`
- `open("file:///...", Connector.READ_WRITE)`
- `openDataOutputStream()`
- `openOutputStream()`
- `openOutputStream(long byteOffset)`

The following `javax.microedition.io.file.FileConnection` methods check for the "write permission":

- `setFileConnection`, when instance opened with WRITE;
- `setFileConnection`, when instance opened with READ_WRITE.

The bottom line prompt in the permission request dialog includes the name of the file or directory only for those protected API calls that have this information specified as a parameter.

The prompt prefix is "<File Location>/<File Name>" for the following methods:

- `open`
- `openDataInputStream`
- `openInputStream`
- `openDataOutputStream`
- `openOutputStream`

File Location represents either:

- "Phone" (when the file is stored on the phone),
  - For example:`open("file:///phone/...").`
- "Card" (when the file is stored on a MMC, SD, T-Flash, or other card-related media)
  - For example:`open("file:///SD/...").`

- "Phone" (when the file is stored on the phone)
  - For example:`open("file:///phone/...")`
- "Card" (when the file is stored on a MMC, SD, T-Flash or other card-related media)
  - For example:`open("file:///SD/...")`

13
JSR-82: Bluetooth API

Overview

JSR-82 covers the establishment of connections between devices for such applications as peer-to-peer gaming and Bluetooth pen use.

There are two new requirements from this API. The `javax.bluetooth` package is required to establish general Bluetooth connections. The `javax.obex` package is required to provide Object Exchange support over Bluetooth and other transports. Because OBEX is not limited to Bluetooth only, it resides as a separate package, but must be supported by this API.

Bluetooth API

The complete requirements are defined in Java APIs for Bluetooth™ Wireless Technology (JSR-82). The requirements listed here are a summary and specify how the API relates to the native Bluetooth implementation on the phone.

System Requirements

The JSR-82 API utilizes Bluetooth for data connections only. The following protocols must be supported:

- L2CAP
- RFCOMM
- SDP
- OBEX
  - OBEX is a separate API from the core Bluetooth API (`javax.bluetooth`) and is a part of the `javax.obex` package.

In addition, the following Bluetooth profiles must be supported:

- Generic Access Profile (GAP)
- Service Discovery Application Profile (SDAP)
- Serial Port Profile (SPP)
- Generic Object Exchange Profile (GOEP)

Bluetooth Control Center

The JSR-82 API requires that a Bluetooth Control Center (BCC) be in place to control the Bluetooth connection and be a repository for local device settings.

According to the API, the following are features the BCC must support:

- A list of remote Bluetooth devices (not necessarily in the vicinity) that are already known to the local Bluetooth device.
- A list of remote Bluetooth devices (not necessarily in the vicinity) that are trusted by the local Bluetooth device.
- A mechanism to bond two devices trying to connect for the first time.
- A mechanism to provide authorization of connection requests.
- The base security settings of the local device, including the security modes defined in the Bluetooth specification.

Device Properties

Table 16 lists the Motorola Bluetooth device properties for current products. These device properties must be available to the MIDlet suite.

Device Property	Description
`bluetooth.api.version`	The version of the Java APIs for Bluetooth™ wireless technology that is supported. For this version, it is set to "1.0".
`bluetooth.l2cap.receiveMTU.max`	The maximum ReceiveMTU size (in bytes) supported in L2CAP. The string is in Base 10 digits, for example, "672." This value is product dependent. The maximum value is 64 Kb.
`bluetooth.connected.devices.max`	The maximum number of connected devices supported (includes parked devices). The string is 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 is 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 is in Base10 digits. This value is product dependent.

Table 16 Motorola Bluetooth Device Properties

Service Registration

Service Registration is the portion of the BCC that controls the Service Discovery Database (SDDB). The SDDB is a list of available services on the local device. Services registered in the SDDB by a MIDlet are removed when the connection notifier is closed or when the MIDlet terminates.

The implementation must support run-before-connect services.

Connectable Mode

The following rules must be supported while the phone is in connectable mode:

Rules:

- In connectable mode, the Bluetooth device periodically listens for connection requests.
- The Bluetooth device responds according to security settings and service availability for requested connection.

Non-Connectable Mode

In non-connectable mode, the Bluetooth device is neither discoverable nor connectable.

Device Management

Device Management describes the local settings involved that control how the local device responds to external requests.

Generic Access Profile (GAP)

These four GAP classes must be supported by the API:

- LocalDevice contains control settings of the local Bluetooth device. Settings can be read and changed.
- RemoteDevice contains information (that is, Bluetooth address and friendly

name) about a remote Bluetooth device.
- DeviceClass contains values of the device type and types of services the device supports.
- BluetoothStateException is an exception that is called when a request cannot be handled because of the device's state.

Security

Security must be set or controlled by the API. Parameters that are available to be set are:

- authentication
- encryption
- authorization
- master (for master/slave switch)

Communication

Communication covers establishing connections to other devices via specific Bluetooth profiles or protocols. Bluetooth connections established using this API are based on the following three protocols:

- RFCOMM
- L2CAP
- OBEX

Additionally, other profiles can be built upon these three basic protocols, but the profiles would have to be emulated by the MIDlet suite.

The implementation must support opening a connection with either a server connection URL or a client connection URL, with the default mode of READ_WRITE.

Serial Port Profile (SPP)

General Rules:

- SPP uses RFCOMM as its protocol.
- Only one RFCOMM session can exist between any pair of devices at any time.
- Negotiation of connection parameters and flow control between two Bluetooth devices must be handled automatically by the SPP connection implementation.
- An SPP server application must initialize the services it offers and register those

services in the SDDB.
- Before an SPP client can establish a connection to an SPP service, it must discover that service via service discovery.
- A service discovery is not required if the SPP service has been discovered previously.

Object Exchange (OBEX)

OBEX is a protocol used for "pushing" and "pulling" objects (that is, files or data) from one device to another. OBEX is not limited to Bluetooth only. OBEX can be used over Bluetooth, IrDA, and USB.

Rules:

- The following OBEX operations *must* be supported by the API:
  - CONNECT
  - PUT
  - GET
  - DISCONNECT
  - SETPATH
  - ABORT
  - CREATE-EMPTY
  - PUT-DELETE
- OBEX *must* support Bluetooth.
- OBEX *may* support the following transports (where available).
  - IrDA
  - TCP/IP
- OBEX *must* support authentication.

Security Policy

Applications *must* be granted permission to perform any requested operation using this API. Table 17 assigns individual permission to the function groups.

Permission	Protocol	Function
`javax.bluetooth`	Bluetooth	Data Networking
`javax.microedition.io.Connector.bluetooth.client`	Bluetooth	Data Networking
`javax.microedition.io.Connector.bluetooth.server`	Bluetooth	Data Networking
`javax.microedition.io.Connector.obey.client`	Bluetooth	Data Networking
`javax.microedition.io.Connector.obex.server`	Bluetooth	Data Networking

Table 17 JSR-82 Security Policy

External Events

The following interruptions must be handled by kvm and MIDlet suite.

Incoming Call

Upon receiving an incoming call, the Bluetooth connection remains active when the MIDlet is suspended. The Bluetooth connection terminates when the user ends the MIDlet.

Incoming Message

Upon receiving an incoming call, the Bluetooth connection remains active when the MIDlet is suspended. The Bluetooth connection is terminated when the user ends the MIDlet.

Alarm and Datebook Behavior

When a MIDlet is running, the Bluetooth connection remains active when the MIDlet is suspended. The Bluetooth connection is terminated when the user ends the MIDlet.

Pressing of End Key

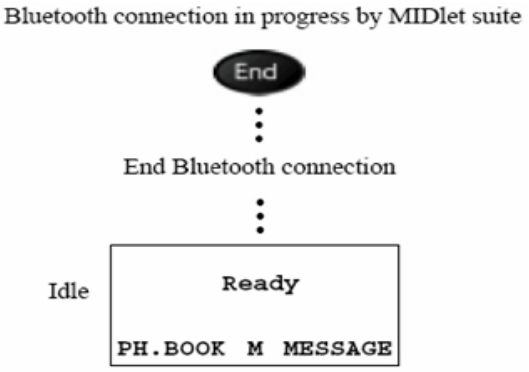


Figure 12 Pressing of End Key

Rules:

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

Hardware Requirements

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

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

Interoperability Requirements

Table 18 lists the suggested types of screens and text used for user feedback.

Examples of each screen type are provided below.

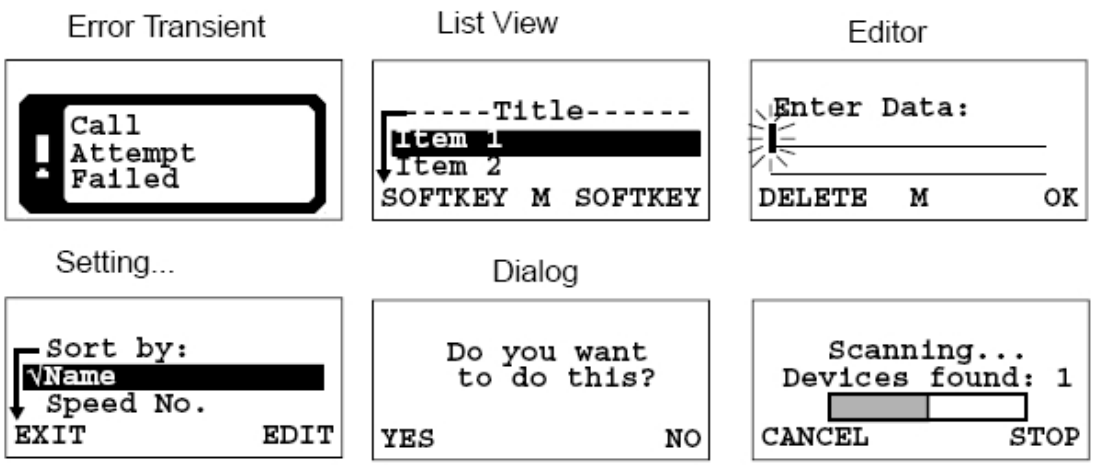


Figure 13 Example Screens

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

tory					Link"
No Devices Found	Transient	"No Devices Found"	N/A	N/A	N/A
New Devices	List	N/A	BACK	LINK	"Scan Results"
PIN Entry	Editor	N/A	DELETE	OK	"Enter PIN"

Table 18 Interoperability Requirements

14 JSR-118: MIDP 2.0 Security Model

Reference	Link
Borland	http://www.borland.com/
GSM 03.38 standard	http://www.etsi.org
GSM 03.40 standard	http://www.etsi.org
IBM	http://www.ibm.com/
MOTODEV	http://developer.motorola.com
Motorola	http://www.motorola.com/
RFC 2068	http://www.ietf.org/rfc/rfc2068.txt
RFC 822	http://www.ietf.org/rfc/rfc822.txt
SAR	http://www.wapforum.org
SSL protocol version 3.0	http://wp.netscape.com/eng/ssl3/ssl-toc.html
Sun Microsystems	http://www.sun.com/
TLS protocol version 1.0	http://www.ietf.org/rfc/rfc2246.txt

This chapter describes the MIDP 2.0 Default Security Model for the MOTOROKR E6/E6e handset. The following topics are discussed:

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

For a detailed MIDP 2.0 Security process diagram, refer to the MOTODEV web site (http://developer.motorola.com).

Table 19 lists the default security feature/class support for MIDP 2.0:

Feature/Class	Implementation
All methods for the Certificate interface in the `javax.microedition.pki package`	Supported
All fields, constructors, methods, and inherited methods for	Supported

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

Table 19 MIDP 2.0 Feature/Class

The domain configuration is selected upon agreement with the operator.

Untrusted MIDlet Suites

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

The following are conditions of untrusted MIDlet suites:

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

Untrusted Domain

Any MIDlet suites that are unsigned belong to the untrusted domain. Untrusted

domain handsets allow, without explicit confirmation, untrusted MIDlet suites access to the following APIs:

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

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

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

Trusted MIDlet Suites

Trusted MIDlet suites are those in which the integrity of the JAR file can be authenticated and trusted by the device, and bound to a protection domain. The MOTOROKR E6/E6e uses x.509PKI for signing and verifying trusted MIDlet suites.

Security for trusted MIDlet suites uses protection domains. Protection domains define permissions that are granted to the MIDlet suite. A MIDlet suite belongs to one protection domain and its defined permissible actions. For implementation on the MOTOROKR E6/E6e, the following protection domains exist:

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

Permissions within these domains authorize access to the protected APIs or functions. These domains consist of a set of "Allowed" or "User" permissions that are granted to the MIDlet suite.

Permission Types Concerning the Handset

A protection domain consists of a set of permissions. Each permission is either "Allowed" or "User":

- "Allowed" (Full Access) permissions explicitly allow access to a given protected API or function from a protected domain. Allowed permissions do not require any user interaction.
- "User" permissions require a prompt to be given to the user and explicit user confirmation to allow the MIDlet suite access to the protected API or function.

User Permission Interaction Mode

User permission for the MOTOROKR E6/E6e handsets allows the user to either deny or grant access to the protected API or function using the following interaction modes (the prompt that appears in bold):

- blanket — grants access to the protected API or function every time it is required by the MIDlet suite until the use uninstalsl the MIDlet suite or changes the permission. (**Ask Once Per App**)
- session — grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is terminated. This mode prompts the user on or before the final invocation of the protected API or function. (**Ask Once Per App**)
- oneshot — prompts the user each time the protected API or function is requested by the MIDlet suite. (**Always Ask**)
- No — does not allow the MIDlet suite access to the requested API or function that is protected. (**No Access**)

The prompt **No, Ask Later** is displayed during run-time dialogs. It allows the user to prohibit access to the protected function this time. However, the next time access is requested, this function is called again.

Security policy and device implementation determine user permission interaction modes. User permission has a default interaction mode. The user is presented with a choice of available interaction modes, including the ability to deny access to the protected API or function. The user makes a decision based on the user-friendly description of the requested permissions provided.

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

Implementation Based on Recommended Security Policy

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

Trusted Third-Party Domain

A trusted third-party protection domain root certificate is used to verify third-party MIDlet suites. These root certificates are mapped to a location on the handset that the user cannot modify. The handset can store a maximum of 12 certificates, consisting of trusted third-party protection domain root certificates and operator protection domain root certificates.

A user can enable a disabled, trusted, third-party, protection domain root certificate. If disabled, the third-party domain is no longer associated with this certificate. Permissions for the trusted third-party domain are "User" permissions. The user grants permissions by responding to a prompt.

Table 20 displays the specific wording for the first line of this prompt:

Protected Functionality	Top Line of Prompt	Right Softkey
Data Network	Use data network?	OK
Messaging	Use messaging?	OK
App Auto-Start	Launch <MIDlet names>?	OK
Connectivity Options	Make a local connection?	OK
User Data Read Capability	Read phonebook data?	OK
User Data Write Capability	Modify phonebook data?	OK
App Data Sharing	Share data between apps?	OK

Table 20 Trusted Third-Party Domain

The following radio button messages, mapped to their corresponding permission types, appear (Table 21):

MIDP 2.0 Permission Types	Run-time Dialogs	UI Permission Prompts
Oneshot	Yes, Always Ask	Always Ask
Session	Yes, Ask Once	Ask Once per App
Blanket	Yes, Always Grant Access	Never Ask
no access	No, Never Grant Access	No Access

Table 21 MIDP 2.0 Permission Types

The run-time dialogs are not displayed if the corresponding permission type is an option for the protected function according to the security policy table loaded into the handset, or when the protected function is set to "Allowed" (or full access).

Security Policy for Protection Domains

Table 22 lists the security policy, by function group, for each domain. Under each domain are the settings allowed for that function; the default setting is in bold. The Function Group appears when the user requests access and when the user modifies the permissions in the menu. The default setting is in effect at the time the MIDlet suite is first invoked and remains in effect until the user changes it.

Permissions are implicitly granted or not granted to a MIDlet based on the configuration of the domain the MIDlet is bound to. Specific permissions cannot be defined for this closed class. A MIDlet either does or doesn't have this capability. The user can change any of the remaining settings.

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

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

Table 22 Security Policy for Protection Domains

Table 23 shows individual permissions assigned to the function groups in Table 23.

MIDP 2.0 Specific Functions		
Permission	**Protocol**	**Function Group**
`javax.microedition.io.Connector.http`	http	Data Network
`javax.microedition.io.Connector.https`	https	Data Network
`javax.microedition.io.Connector.datagram`	datagram	Data Network
`javax.microedition.io.Connector.datagramreceiver`	datagram server (w/o host)	Data Network
`javax.microedition.io.Connector.socket`	socket	Data Network
`javax.microedition.io.Connector.serversocket`	server socket (w/ o host)	Data Network
`javax.microedition.io.Connector.ssl`	ssl	Data Network
`javax.microedition.io.Connector.comm`	comm	Connectivity Options
`javax.microedition.io.PushRegistry`	All	App Auto-Start
Wireless Messaging API - JSR-120		
`javax.wireless.messaging.sms.send`		Messaging
`javax.wireless.messaging.sms.receive`		Messaging
`javax.microedition.io.Connector.sms`		Messaging
`javax.wireless.messaging.cbs.receive`		Messaging
Multimedia Recording		
`javax.microedition.media.RecordControl.startRecord`	`RecordControl.startRecord ( )`	Multimedia Recording

Table 23 MIDP 2.0 Specific Functions

Each phone call or messaging action lets the user see the destination phone number before approving the action. The handset ensures that I/O access from the Mobile Media API follows the same security requirements as the Generic Connection Framework.

Displaying Permissions

Permissions are divided into function groups, which the user can view. Each function group falls into one of two categories: Network/Cost related and User/Privacy related.

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

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

The user can access and modify only third-party and untrusted permissions. Operator and manufacturer permissions are displayed for each MIDlet suite, but the user cannot modify them.

Trusted MIDlet Suites Using x.509 PKI

Using the x.509 PKI (Public Key Infrastructure) mechanism, the handset can verify the signer of the MIDlet suite and bind it to a protection domain that allows the MIDlet suite access to the protected API or function. When the MIDlet suite is bound to a protection domain, it uses the permission defined in the protection domain to grant the MIDlet suite access to the defined protected APIs or functions.

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

Signing a MIDlet Suite

The default security model involves the MIDlet suite, the signer, and public key certificates. A set of root certificates are used to verify certificates generated by the signer. Specially designed certificates for code signing can be obtained from the manufacturer, operator, or certificate authority. The MOTOROKR E6/E6e handset supports only those root certificates that are stored on the handset.

Signer of MIDlet Suites

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

MIDlet Attributes Used in Signing MIDlet Suites

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

is not installed.

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

Methods:

- `MIDlet.getAppProperty` returns the attribute value, if present, from the manifest (JAR). If an attribute value is not defined, the attribute value, if present, is returned from the application descriptor (JAD).

Creating the Signing Certificate

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

Inserting Certificates into JAD

When inserting a certificate into a JAD, the certificate path includes the signer certificate and any other necessary certificates. It omits the root certificate, which is found on the device only.

Each certificate is encoded using base64 without line breaks, and inserted into the application descriptor as outlined below per MIDP 2.0.

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

Note the following:

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

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

Creating the RSA SHA-1 Signature of the JAR

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

The signer of the MIDlet suite is responsible for its protection domain root certificate owner for protecting the domain's APIs and protected functions; therefore, the signer checks the MIDlet suite before signing it. Protection domain root certificate owners can delegate the signing of MIDlet suites to a third party and in some instances, the author of the MIDlet.

Authenticating a MIDlet Suite

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

Verifying the Signer Certificate

The signer certificate is found in the application descriptor of the MIDlet suite. The process for verifying a signer certificate is as follows:

1. Get the certification path for the signer certificate from the JAD attributes MIDlet-Certificate-1<*m*>, where <*m*> starts at 1 and is incremented by 1 until there is no attribute with this name. The value of each attribute is a base64 encoded certificate that needs to be decoded and parsed.
2. Validate the certification path using the basic validation process as described in RFC 2459 using the protection domains as the source of the protection domain root certificates.
3. Bind the MIDlet suite to the corresponding protection domain that contains the protection domain root certificate that validated the first chain from signer to root.
4. Begin installation of MIDlet suite.
5. If attribute MIDlet-Certificate-<*n*>-<*m*> where <*n*> is greater than 1, is present and full certification path is not established after verifying MIDlet-Certificate-<1>-<*m*> certificates, then repeat steps 1 through 3 for the value <*n*> greater by 1 than the previous value.

Table 24 describes actions performed upon completion of signer certificate verification:

Result	Action
Attempted to validate <*n*> paths. However, issuer's public keys are missing or certificate pathsa are invalid.	Authentication fails, JAR installation is not allowed.
More than one full certification path is established and validated.	Implementation proceeds with the signature verification using the first successfully verified certificate path for authentication and authorization.
Only one certification path is established and validated.	Implementation proceeds with the signature verification.

Table 24 Actions Performed of Signer Certificate Verification

Verifying the MIDlet Suite JAR

To verify the MIDlet suite JAR:

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

Table 25 is a summary of MIDlet suite verification including messages:

Initial State	Verification Result
JAD not present, JAR downloaded	Authentication cannot be performed, install JAR. MIDlet suite is treated as untrusted. The following error message appears: "Application installed, but may have limited functionality."
JAD present, but JAR is unsigned	Authentication cannot be performed, install JAR. MIDlet suite is treated as untrusted. The following error message appears: "Application installed, but may have limited functionality."
JAR signed but no root certificate present in the keystore to validate the certificate chain	Authentication cannot be performed. JAR installation is not allowed. The following error message appears: "Root certificate missing. Application not installed."
JAR signed, a certificate on the path is expired	Authentication cannot be completed. JAR installation is not allowed. The following error message appears: "Expired Certificate. Application not installed."
JAR signed, a certificate rejected for reasons other than expiration	JAD rejected, JAR installation is not allowed. The following error message appears: "Authentication Error. Application not installed."
JAR signed, certificate path validated but signature verification fails	JAD rejected, JAR installation is not allowed. The following error message appears: "Authentication Error. Application not installed."
Parsing of security attributes in JAD fails	JAD rejected, JAR installation is not allowed. The following error message appears: "Failed Invalid File."
JAR signed, certificate path	JAR is installed. The following message appears: "In-

validated, signature verified	stalled."

Table 25 Summary of MIDlet Suite Verification

Carrier Specific Security Model

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

15 JSR-120: Wireless Messaging API

Wireless Messaging API (WMA)

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

The JSR-120 specification states that developers send (MO - mobile originated) and receive (MT - mobile terminated) SMS (Short Message Service) on the target device.

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

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

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

SMS Client Mode and Server Mode Connection

The Wireless Messaging API is based on the Generic Connection Framework (GCF), which is defined in the CLDC specification 1.1. The use of the "Connection" framework in Motorola's case is " `MessageConnection`".

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

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

Messages received with this identifier are then 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 that points to another device. A client mode connection can only be used for sending messages.

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

SMS Port Numbers

When a port number is present in the address, the TP-User-Data of the SMS 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 Java ME MIDlet cannot receive this kind of message, but the SMS is 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 is received by this `MessageConnection`.

However, when a client type `MessageConnection` is used for sending a message with

a port number, the originating port number is set to an implementation specific value and any possible messages received to this port number are not delivered to the `MessageConnection`. Please refer to the sections A.4.0 and A.6.0 of the JSR-120.

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

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

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

SMS Storing and Deleting Received Messages

When SMS messages are received by the MIDlet, they are removed from the SIM card memory where they were stored. The storage location (inbox) for the SMS messages has a capacity of up to thirty messages. If any messages are older than five days, then they are 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 is 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 supports at least three SMS messages to be received and concatenated together. In addition, a minimum of three messages for sending is supported. Motorola advises that developers should not send messages that take up more than three SMS protocol messages unless the recipient's device is known to support more.

SMS Notification

Examples of SMS interaction with a MIDlet include:

- A MIDlet handles an incoming SMS message if the MIDlet is registered to receive messages on the port (identifier) and is running.
- When a MIDlet is paused and is registered to receive messages on the port number of the incoming message, then the user is queried to launch the MIDlet.
- If the MIDlet is not running and the Java Virtual Machine is not initialized, then a Push Registry is used to initialize the Virtual Machine and launch the Java ME MIDlet. This only applies to trusted, signed MIDlets.

- If a message is received and the untrusted unsigned application and KVM are not running, then the message is 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 26 is a list of Messaging features/classes supported in the device.

Feature/Class	Implementation
JSR-120 API — APIs defined in the javax.wireless.messaging package are implemented with regards to the GSM SMS Adaptor	Supported
Removal of SMS messages	Supported
Terminated SMS removal — any user prompts handled by MIDlet	Supported
Originated SMS removal — any user prompts handled by MIDlet	Supported
All fields, methods, and inherited methods for the Connector Class in the `javax.microedition.io` package	Supported
All methods for the BinaryMessage interface in the `javax.wireless.messaging` package	Supported
All methods for the Message interface in the `javax.wireless.messaging` package	Supported
All fields, methods, and inherited methods for the MessageConnection interface in the `javax.wireless.messaging` package	Supported
Number of MessageConnection instances in the `javax.wireless.messaging` package	32 maximum
Number of MessageConnection instances in the `javax.wireless.messaging` package	16
All methods for the MessageListener interface in the `javax.wireless.messaging` package	Supported
All methods and inherited methods for the TextMessage interface in the `javax.wireless.messaging` package	Supported
16-bit reference number in concatenated messages	Supported
Number of concatenated messages	30 messages in inbox, each can be concatenated from 3 parts. No limitation on outbox (immediately transmitted)

Allow MIDlets to obtain the SMSC address with the `wireless.messaging.sms.smsc` system property	Supported

Table 26 List of Messaging Features/Classes

Code Sample 5 shows implementation of the JSR-120 Wireless Messaging API.

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

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

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

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

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

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

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

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

Creation of server connection MessageConnection messageConnection = (MessageConnection)Connector.open("sms://:9532");
Creation of client connection with port number MessageConnection messageConnection = (MessageConnection) Connector.open("sms://+18473297274:9532");
Creation of client connection without port number MessageConnection messageConnection = (MessageConnection)Connector.open("sms://+18473297274");
Closing of connection MessageConnection messageConnection.close();
Creation of SMS message Message textMessage = messageConnection.newMessage(MessageConnection. TEXT_MESSAGE);
Setting of payload text for text message ((TextMessage)message).setPayloadText("Text Message");
Getting of payload text of received text message receivedText = ((TextMessage)receivedMessage).getPayloadText();
Getting of payload data of received binary message BinaryMessage binMsg; byte[] payloadData = binMsg.getPayloadData();
Setting of address with port number message.setAddress("sms://+18473297274:9532");
Setting of address without port number message.setAddress("sms://+18473297274");

Sending of message

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

Receiving of message

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

Getting of address

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

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

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

Getting of timestamp for the message

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

Setting of MessageListener and receiving of notifications about incoming messages

```
public class JSR120Sample1 extends MIDlet implements CommandListener {

JSR120Sample1Listener listener = new JSR120Sample1Listener();

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

// create message to send

listener.run();

// set payload for the message to send

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

// send message (via invocation of 'send' method)

// set address for the message to receive
receivedMessage.setAddress("sms://:9532");
```

```
// receive message (via invocation of 'receive' method)

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

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

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

Code Sample 5 JSR-120 WMA

16 JSR-135: Mobile Media API

Mobile Media API

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

- Tone Sequence
- Sampled Audio
- MIDI
- Interactive MIDI

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

Code Sample 5 shows the implementation of the JSR-135 Mobile Media API:

JSR-135

```
                                    Player player;

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

```
                                        }
                                        // Obtain the information required to acquire
the media resources try
                                        {
                                                player.realize();
                                        }
                                        catch (MediaException e)
                                        {
                                                System.out.println("FAILED: exception
for realize: " + e.toString());
                                        }
                                        // Acquire exclusive resources, fill buffers
with media data try
                                        {
                                                player.prefetch();
                                        }
                                        catch (MediaException e)
                                        {
                                                System.out.println("FAILED: exception
for prefetch: " + e.toString());
                                        }
                                        // Start the media playback try
                                        {
                                                player.start();
                                        }
                                        catch (MediaException e)
                                        {
                                                System.out.println("FAILED: exception
for start: " + e.toString());
                                        }
                                        // Pause the media playback try
                                        {
                                                player.stop();
                                        }
                                        catch (MediaException e)
                                        {
                                                System.out.println("FAILED: exception
for stop: " + e.toString());
                                        }
                                        // Release the resources
                                                                                play-
er.close();
```

Code Sample 5 JSR-135 MMA

ToneControl

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

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

The following is the available method for ToneControl:

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

VolumeControl

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

The following describes the different volume settings found within VolumeControl:

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

The following is a list of available methods for VoumeControl:

`-getLevel`: Gets the current volume setting.

`-isMuted`: 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 public interface StopTimeControl.

The following is a list of available methods for StopTimeControl:

`-getStopTime`: Gets the last value successfully set 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 a 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

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

Audio Media

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

Table 26 Audio Media

Image Media

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

Video Media

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

Media Locators

The Manager and the DataSource class, as well as the RecordControl interface accept media locators. In addition to normal playback locators specified by JSR-135, the following special locators need to be supported.

RTSP locator

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

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

HTTP Locator

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

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

Capture Locator

Capture Locator should be supported for audio and video devices. A new device "camera" must be defined and supported for camera device. The `Manager.createPlayer()` method returns camera player as a special type of video player. Camera player should implement VideoControl and should support taking snapshots using the `VideoControl.getSnapshot()` method.

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

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

Security

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

Policy3

The following is flexed in per operator requirements at ship time of the handset.

Function Group	Trusted Third Party	Untrusted	Manufacturer	Operator
Multimedia Record	**Ask once Per App**, Always Ask, Never Ask, No Access	**Always Ask**, Ask Once Per App, Never Ask, No Acess	Full Access	Full Access

Permissions

Table 31 lists individual permissions within Multimedia Record function group.

Permission	Protocol	Function Group
`javax.microedition.media.control.RecordControl.re`	`RecordCon-trol.startRecord()`	MultimediaRecord

Table 31 Permissions within Multimedia Record

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

17
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. An explanation of CLDC 1.0 follows.

JSR-30 — CLDC 1.0

This CLDC Specification addresses the following areas:

- Java language and virtual machine features
- Core Java libraries `(java.lang.*, java.util.*)`
- Input/output
- Networking
- Security
- Internationalization

This CLDC Specification does not address the following features:

- Application life-cycle management (application installation, launching, deletion)
- User interface functionality
- Event handling
- High-level application model (the interaction between the user and the application)

No Floating Point Support

The main language-level difference between the full Java™ Language Specification and this CLDC Specification is that a JVM supporting CLDC does not have floating point support.

This means that a JVM supporting CLDC does not allow the use of floating point literals, floating point types and values, and floating point operations.

Classfile Format and Class Loading

An essential requirement for the Connected, Limited Device Configuration is the ability to support dynamic downloading of Java applications and Third party content. The dynamic class loading mechanism of the Java platform plays a central role in enabling this. This section discusses the application representation formats and class loading practices required of a JVM supporting CLDC.

Supported File Formats

It is assumed that a CLDC implementation is able to read standard Java class files with the pre-verification changes defined in the following section "Public Representation of Java Applications and Resources." In addition, a CLDC implementation supports compressed Java Archive (JAR) files. This requirement has been added to maintain upward compatibility with larger Java environments and existing Java tools, but with a smaller footprint than with regular class files. Detailed information about JAR format is provided at *http://java.sun.com/developer/Books/javaprogramming/JAR/index.html.*

Public representation of Java applications and resources

A Java application is considered to be "represented publicly" or "distributed publicly" when the system it is stored on is open to the public, and the transport layers and protocols it can be accessed with are open standards. In contrast, a device can be part of a single, closed network system where the vendor controls all communication. In this case, the application is no longer represented publicly once it enters and is distributed via the closed network system. Whenever Java applications intended for a CLDC device are represented publicly, the compressed JAR file representation format must be used. The JAR file must contain regular Java class files with the following restrictions and additional requirements:

- stack map attributes must be included in class files.
- the class file must not contain any of the following Java byte codes: jsr, jsr_w, ret and wide ret.

Sun's CLDC reference implementation includes a pre-verification tool for performing the above modifications to a Java class file. The stack map attributes are automatically ignored by the conventional class file verifier, that is, the format specified here is fully upwards compatible with larger Java environments such as Java SE or Java EE.

Additionally, the JAR file may contain application-specific resource files that can be loaded into the virtual machine by calling on of the following methods:

- `Class.getResourceAsStream(String name)`
- `getClass().getResourceAsStream(String name)`

Classfile Lookup Order

The Java™ Language Specification and Java™ Virtual Machine Specification do not specify the order in which class files are searched when new class files are loaded into the virtual machine. At the implementation level, a typical Java virtual machine implementation utilizes a special environment variable classpath to define the lookup order.

This CLDC Specification assumes class file lookup order to be implementation-dependent, with the following restrictions. The lookup strategy is typically defined as part of the application management implementation. A JVM supporting CLDC is not required to support the notion of classpath, but may do so at the implementation level. Two restrictions apply to class file lookup order.

- First, "Protecting system classes," a JVM supporting CLDC must guarantee that the application programmer cannot override the system classes (classes belonging to the CLDC or supported profiles) in any way.
- Second, it is required that the application programmer must not be able to manipulate the class file lookup order in any way. Both of these restrictions are important for security reasons.

JSR-139 — CLDC 1.1

The Implementation of CLDC 1.1 supports the following:

- Floating Point
  - Data Types float and double

    - New Data Type classes Float and Double
    - All floating point byte codes
    - Library classes to handle floating point values
- Weak reference
- Classes Calender, Date, and TimeZone are Java SE compliant
- Thread objects are compliant with Java SE

The support of thread objects to be compliant with Java SE requires the addition of `Thread.getName` and a few new constructors. The following table lists the additional classes, fields, and methods supported for CLDC 1.1 compliance:

	Classes	Additional Fields/ Methods	Comments
System Classes	`java.lang.Thread`	`Thread (Runnable target, String name)`	Allocates a new Thread object with the given target and name.
		`Thread (String name)`	Allocates a new Thread object with the given name.
		`String getName ()`	Returns this thread's name.
		`Void interrupt ()`	Interrupts this thread.
	`java.lang.String`	`Boolean equalIgnoreCase (String anotherString)`	Compares this string to another String, ignoring case considerations.
		`String intern ()`	Returns a canonical representation for the string object.
		`static String valueOf (float f)`	Returns the string representation of the float argument.
		`static String valueOf (double d)`	Returns the string representation of the double argument.
Data Type Classes	`java.lang.Float`		New Class: Refer to CLDC Spec for more details.
	`java.lang.Double`		New Class: Refer to CLDC Spec for more details.
Calendar and Time Classes	`java.util.Calendar`	`protected int [ ] fields`	The field values for the currently set time for

			this calendar.
		protected boolean { } is set	The flags that tell if a specified time field for the calendar is set.
		Protected long time	The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT.
		protected abstract void ComputeFields	Converts the current millisecond time value to field values in fields [ ].
		protected abstract void ComputeTime	Converts the current field values in fields [ ] to the millisecond value time.
	java.lang.Date	String toString ()	Converts this date object to a String of the form: Dow mon dd hh:mm:ss zzz yyyy.
Exception and Error Classes	java.lang.NoClassDefFoundError		New Class: Refer to CLDC Spec for more details.
Weak References	java.lang.ref.Reference		New Class: Refer to CLDC Spec for more details.
	java.lang.ref.WeakReference		New Class: Refer to CLDC Spec for more details.
Additional Utility Classes	java.util.Random	double nextDouble ()	Returns the next pseudo-random, uniformly distributed double value between 0.0 and 1.0 from the random number generator's sequence.
		float nextFloat()	Returns the next pseudo-random, uniformly distributed double value between 0.0 and 1.0 from the random number generator's sequence.
		int nextInt (int	Returns a pseudo-

		n)	random, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
	java.lang.Math	static double E	The double value that is closer than any other to e, the base of the natural logarithms.
		static double PI	The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter.
		static double abs (double a)	Returns the absolute value of a double.
		static float abs (float a)	Returns the absolute value of a float value.
		static double ceil (double a)	Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.
		static double cos (double a)	Returns the trigonometric cosine of an angle.
		static double floor (double a)	Returns the largest double value that is not greater than the argument and is equal to a mathematical integer.
		static double max (double a, double b)	Returns the greater of two double values.
		static float max (float a, float b)	Returns the greater of two float values.
		static double min (float a, float	Returns the smaller of two double values.

		b)	
		static float min (float a, float b)	Returns the smaller of two float values.
		static double sin (double a)	Returns the trigono-metric sine of an angle.
		static double sqrt (double a)	Returns the correctly rounded positive square root of a double value.
		static double tan (double a)	Returns the trigono-metric tangent of angle.
		static double to-degrees (double angrad)	Converts an angle measured in radians to the equivalent angle measured in degrees.
		static double to-radians (double angdeg)	Converts an angle measured in degrees to the equivalent angle measured in radians.

Table 31 Additional Classes, Fields, and Methods Supported for CLDC 1.1 Compliance

18 JSR-172: Java™ ME Web Services Specification

This chapter describes the JSR-172, which uses the Web Services standards and infrastructures to provide the programming model for the next generation of enterprise services. Two new capabilities are provided here for the Java™ ME plataform:

- Access to remote SOAP / XML based web services
- Parsing XML data

Overview

The main deliverables of the JSR-172 specification are two, independent, optimal packages:

1. An optional package adding XML Parsing support to the platform. Structured data sent to mobile devices from existing applications will likely be in the form of XML. To avoid including code to process this data in each application, it is desirable to define an optional package that can be included with the platform.
2. Create an optional package to facilitate access to XML based web services from CDC and CLDC based profiles.

This optional package defines an API to allow mobile devices to access remote XML based web services. Where possible, it avoids defining new network protocols and formats and reuses existing standards.

NOTE: Java™ ME Optional Packages are described in JSR-68, Java™ ME Platform Specification.

JAXP

The goal of this optional package is to define a strict subset wherever possible of the XML parsing functionality (defined in JSR-063 JAXP 1.2) that can be used on the Java™ Platform, Micro Edition (Java™ ME).

XML is becoming a standard means for clients to interact with backend servers, their databases, and related services. With its platform neutrality and strong industry support, XML is being used by developers to link networked clients with remote enterprise data. An increasing number of these clients are based on the Java™ ME platform, with a broad selection of mobile phones, PDAs, and other portable devices. As developers utilize these mobile devices more to access remote enterprise data, XML support on the Java™ ME platform is becoming a requirement.

An implementation *may* support validation of XML documents against a DTD. XML validation is an expensive process in terms of processing power and memory usage and would not likely be supported on most Java™ ME devices. However, if the platform has the ability to support it, it *may* provide a validating parser (due to the limited nature of most Java™ ME devices, it is expected that only one parser will be supported, but it is allowable to support both).

There are three packages that comprise the JAXP API subset:

- `javax.xml.parsers`
- `org.xml.sax`
- `org.xml.sax.helpers`

When inspecting the API set, one quickly notices that much of what exists in the Java™ SE JAXP API set is missing from the Java™ ME JAXP API set. The size requirements for the Java™ ME platform are strict, allowing only approximately 35Kb for a complete JAXP implementation. However, although many of the classes are gone, much of the functionality remains.

JAX-RPC Subset Overview

JAX-RPC is a Java API for interacting with SOAP based web services. This specification defines a subset of the JAX-RPC 1.1 specification that is appropriate for the Java™ ME platform.

The functionality provided in the subset reflects both the limitations of the platform: memory size and processing power; as well as the limitations of the deployment environment: low bandwidth and high latency.

Implementations *must* support WSDL documents, referencing the following data types:

- boolean
- byte
- short
- int
- long
- float
- double
- String
- complex types
- arrays of primitive and complex types

The following classes and interfaces are included in the Java™ ME Web Services Optional Package to satisfy dependencies of JAX-RPC on the CLDC based platforms:

- `java.rmi.Remote`
- `java.rmi.RemoteException`
- `java.rmi.MarshalException`
- `java.rmi.ServerException`

An RMI Optional Package is available for CDC based platforms, and if the optional package is present, the versions of `java.rmi.Remote`, `java.rmi.RemoteException`, `java.rmi.MarshalException`, and `java.rmi.ServerException` included in the RMI optional package *must* be used.

19 JSR-184: Mobile 3D Graphics API

Overview

JSR-184 Mobile 3D API defines an API for rendering three-dimensional (3D) graphics at interactive frame rates, including a scene graph structure and a corresponding file format for efficient management and deployment of 3D content. Typical applications that might make use of JSR-184 Mobile 3D API include games, map visualizations, user interface, animated messages, and screen savers. JSR-184 requires a Java™ ME device supporting MIDP 2.0 and CLDC 1.1 at a minimum.

Mobile 3D API

The MOTOROKR E6/E6e contains full implementation of JSR-184 Mobile 3D API (*http://jcp.org/en/jsr/detail?id=184*). The MOTOROKR E6/E6e has also implemented the following:

- Call to `System.getProperty` with key — `microedition.m3g.version` returns 1.0, otherwise NULL is returned.
- Floating point format for input and output is the standard IEEE float, having an 8-bit exponent and a 24-bit mantissa normalized to 1.0, 2.0.
- Implementation ensures the Object3D instances are kept in reference to reduce overhead and possible inconsistency.
- Thread safety.
- Necessary pixel format conversions for rendering output onto device.
- Support at least 10 animation tracks to be associated with an Object 3D

instance (including animation controller), subject to dynamic memory availability.

Mobile 3D File Format Support

The MOTOROKR E6/E6e supports both M3G and PNG file formats for loading 3D content. The MOTOROKR E6/E6e supports the standard .m3g and .png extensions for its file formats. MIME type and not extension is used for identifying file type. In the case that the MIME type is not available, M3G files are identified using the file identifier and PNG files using the signature.

Mobile 3D Graphics — M3G API

The M3G API lets you access the realtime 3D engine embedded on the device to create console quality 3D applications, such as games and menu systems. The main benefits of the M3G engine include:

- The whole 3D scene can be stored in a very small file size (typically 50-150K), allowing you to create games and applications in under 256K;
- The application can change the properties (such as position, rotation, scale, color, and textures) of objects in the scene based on user interaction with the device;
- The application can switch between cameras to get different views in to the scene;
- The rendered images have a very high photorealistic quality.

Typical M3G Application

An application consists of logic that uses the M3G, MIDP 2.0, and CDLC 1.1 classes. The application is compiled into a Java MIDlet that can be embedded on the target device. The MIDlet can also contain additional assets, such as one or more M3G files that define the 3D scene graph for the objects in the scene, images, and sounds.

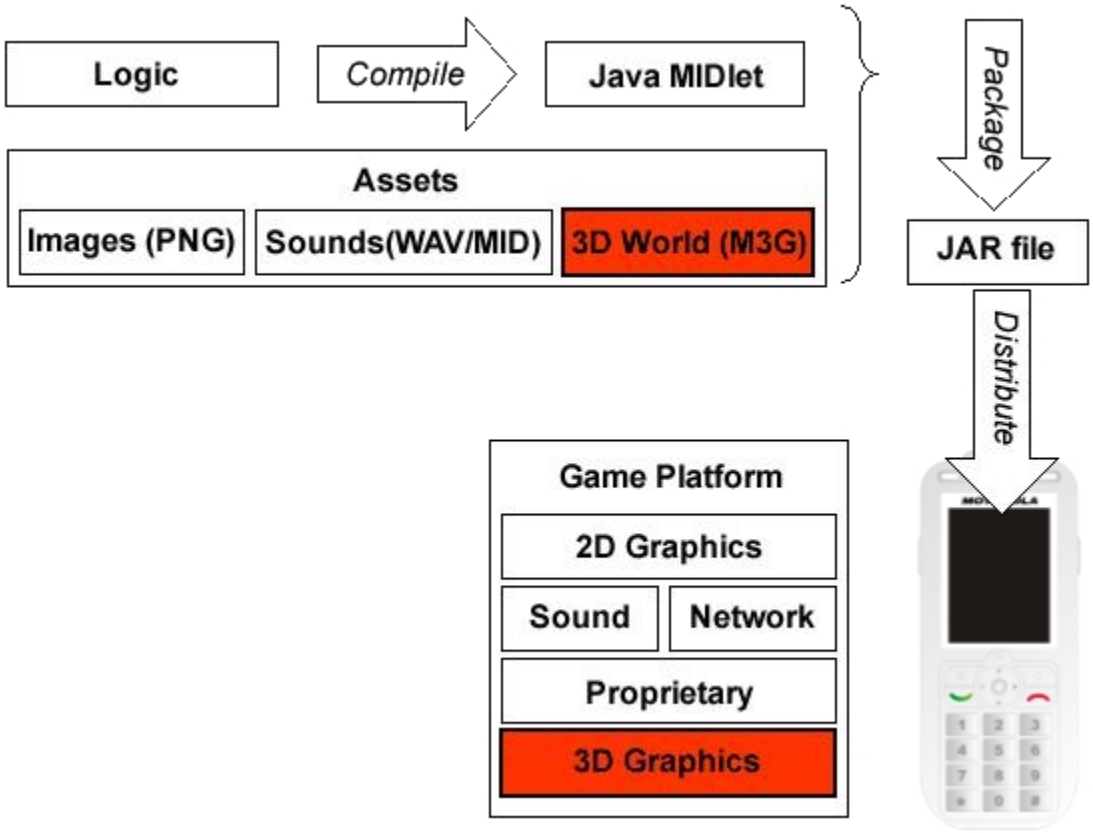


Figure 14 M3G Application Proccess

Most M3G applications use an M3G resource file that contains all the information required to define the 3D resources, such as objects, their appearance, lights, cameras, and animations in a scene graph. The file must be loaded into memory where object properties can be interrogated and altered using the M3G API. Alternatively, all objects can be created from code, although this is likely to be slower and limits creativity for designers.

Simple MIDlets

The simplest application consists of an M3G file that is loaded into the application using the M3G Loader class, which is then passed to a Graphics3D object that renders the world to the Display.

Figure 15 M3G Application Methods

The World object contains the objects that define a complete 3D scene — geometry, textures, lights, cameras, and animations. The World object mediates access to the objects within the world. It can be passed as a block to the renderer, the Graphics3D class.

The Loader object, populates a World by loading an M3G file from a URI or other asset source, such as a buffer of bytes in M3G format. The Loader is not restricted to loading just Worlds, each file can contain as little as a single object and multiple files can be merged together on the device, or you can put everything into a single file.

The rendering Graphics3D class (by analogy to the MIDP Graphics class) takes a whole scene (or part of a scene graph), and renders a view onto that scene using the current camera and lighting setup. This view can be to the screen, to a MIDP image, or to a texture in the scene for special effects. You can pass a whole world in one go (retained mode) or you can pass individual objects (immediate mode). There is only one Graphics3D object present at one time, so that hardware accelerators can be used.

Figure 16 shows the structure of a more typical MIDlet.

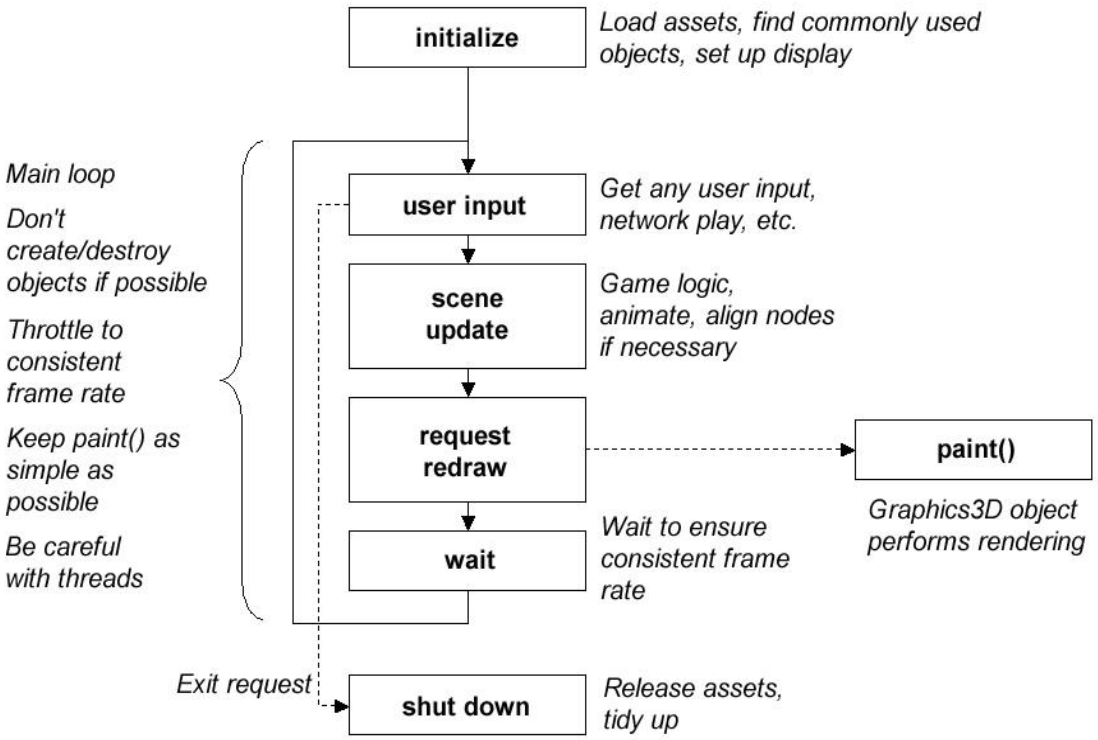


Figure 16 Typical MIDlet Structure

Initializing the World

The Loader class is used to initialize the world. It has two static methods: one takes in a byte array, while the other takes a named resource, such as a URI or an individual file in the JAR package.

The load methods return an array of Object3Ds that are the root level objects in the file.

The following example calls `Loader.load()` and passes it an M3G file from the JAR file using a property in the JAD file. Alternatively, you could specify a URI. For example:

```
Object3D[] roots = Loader.load(http://www.example.com/m3g/
simple.m3g)[0];
```

The example assumes that there is only one root node in the scene, which is the world object. If the M3G file has multiple root nodes, the code must be changed to

reflect this, but generally most M3G files have a single root node.

Initializing the World

```
public void startApp() throws MIDletStateChangeException
{
	myDisplay.setCurrent(myCanvas);

	try
	{
		// Load a file.
		Objects3D[] roots = Loader.load(getAppProperty("Content-1"));

		// Assume the world is the first root node loaded.
		myWorld = (World) roots[0];
	}
	catch(Exception e)
	{
		e.printStackTrace();
	}

	// Force a repaint so the update loop is started.
	myCanvas.repaint();
}
```

Code Sample 5 Initializing the World

Using the Graphics3D Object

Using the Graphics3D class is very straightforward. Get the Graphics3D instance, bind a target to it, render everything, and release the target.

Using the Graphics3D Object

```
public class myCanvas extends Canvas
{
	Graphics3D myG3D = Graphics3D.getInstance();

	public void paint(Graphics g)
	{
```

```
            myG3D.bindTarget(g);

            try
            {
                  myG3D.render(myWorld);
            }
            finally
            {
                  myG3D.releaseTarget();
            }

      }

}
```

Code Sample 8 Using the Graphics3D Object

The final block makes sure that the target is released and the Graphics3D object can be reused. The `bindTarget` call must be outside the try block, as it can throw exceptions that cause `releaseTarget` to be called when a target has not been bound.

Interrogating and Interacting with Objects

The World object is a container that sits at the top of the hierarchy of objects that form the scene graph. You can find particular objects within the scene very simply by calling `find()` with an ID. The `find()` method returns a reference to the object that has been assigned that ID in the authoring tool (or manually assigned from code). This is important because it makes the application logic independent of the detailed structure of the scene.

Finding Objects by ID

```
final int PERSON_OBJECT_ID = 339929883;
Node personNode = (Node)theWorld.find(PERSON_OBJECT_ID);
```

Code Sample 9 Finding Objects by ID

If you need to find many objects, or you don't have a fixed ID, then you can follow the hierarchy explicitly using the `Object3D.getReferences()` or `Group.getChild()`

methods.

Using the Object3D.getReferences()

```
static void traverseDescendants(Object3D obj)
{
	int numReferences = obj.getReferences(null);

	if (numReferences > 0)
	{
		Object3D[] objArray = new Object3D[numReferences];

		obj.getReferences(objArray);

		for (int i = 0; i < numReferences; i++)
			traverseDescendants(objArray[i]);

	}

}
```

Code Sample 10 Using the Object3D.getReferences()

Once you have an object, most of the properties on it can be modified using the M3G API. For example, you can change the position, size, orientation, color, brightness, or whatever other attribute of the object is important. You can also create and delete objects and insert them into the world, or link parts of other M3G files into the scene graph.

Animations

As well as controlling objects from code, scene designers can specify how objects should move under certain circumstances, and store this movement in 'canned' or block animation sequences that can be triggered from code. Many object properties are animatable, including position, scale, orientation, color and textures. Each of these properties can be attached to a sequence of keyframes using an

AnimationTrack. The keyframe sequence can be looped, or just played once, and they can be interpolated in several ways (stepwise, linear, spline).

A coherent action typically requires the simultaneous animation of several properties on several objects, the tracks are grouped together using the AnimationController object. This allows the application to control a whole animation from one place.

All the currently active animatable properties can be updated by calling `animate()` on the World. (You can also call this on individual objects if you need more control.) The current time is passed through to `animate()` and is used to determine the interpolated value to assign to the properties.

The `animate()` method returns a validity value that indicates how long the current value of a property is valid. Generally, this is 0 which means that the object is still being animated and the property value is no longer valid, or infinity when the object is in a static state and does not need to be updated. If nothing is happening in the scene, you do not have to continually redraw the screen, reducing the processor load and extending battery life. Similarly, simple scenes on powerful hardware may run very fast; by restricting the frame-rate to something reasonable, you can extend battery life and are more friendly to background processes.

The animation subsystem has no memory, so time is completely arbitrary. This means that there are no events reported (for example, animation finished). The application is responsible for specifying when the animation is active and from which position in the keyframe sequence the animated property is played.

Consider a world, myWorld, that contains an animation of 2000 ms that you want to cycle. First, you need to set up the active interval for the animation and set the position of the sequence to the start. Then call `World.animate()` with the current world time:

Setting animation interval

```
anim.setActiveInterval(worldTime, worldTime+2000);
anim.setPosition(0, worldTime);

int validity = myWorld.animate(worldTime);
```

Code Sample 11 Setting animation interval

Authoring M3G files

You can create all your M3G content from code if necessary but this is likely to be very time consuming and does not allow 3D artists and scene designers to easily create and rework visually compelling content with complex animations. You can use professional, visual development tools such as Swerve™ Studio or Swerve™ M3G exporter from Superscape Group, which exports content from 3ds max, the industry standard 3D animation tool, in fully compliant M3G format. For more information, please visit *http://www.superscape.com*

20 JSR-185: Java Technology for the Wireless Industry

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

Overview

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

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

Any Motorola JTWI device implements the following and passes the corresponding TCK:

- CLDC 1.0 or CLDC 1.1
- MIDP 2.0 (JSR-118)
- Wireless Messaging API 1.1 (JSR-120)
- Mobile Media API 1.1 (JSR-135)

CLDC Related Content for JTWI

JTWI is designed to be implemented on top of CLDC 1.0 or CLDC 1.1. The configuration provides the VM and the basic APIs of the application environment. If floating point capabilities are exposed to Java Applications, CLDC 1.1 is implemented.

The following CLDC requirements are supported:

- Minimum Application thread count allows a MIDlet suite to create a minimum of 10 simultaneous running threads
- Minimum Clock Resolution — The `java.land.System.currentTimeMillis()` method records the elapsed time in increments not to exceed 40 msec. At least 80% of test attempts will meet the time elapsed requirement to achieve acceptable conformance.
- Names for Encodings support at least the preferred MIME name as defined by IANA (*http://www.iana.org/assignments/character-sets*) for the supported character encodings. If preferred name has not been defined, the registered name is used (that is, UTF-16).
- Character Properties provide support for character properties and case conversions for the characters in the Basic Latin and Latin-1 Supplement blocks of Unicode 3.0. Other Unicode character blocks are supported as necessary.
- Unicode Version supports Unicode characters. Character information is based on the Unicode Standard version 3.0. Since the full character tables required for Unicode support can be excessively large for devices with tight memory budgets, by default, the character property and case conversion facilities in CLDC assume the presence of ISO Latin-1 range of characters only. Refer to JSR-185 for more information.
- Custom Time Zone IDs permit the use of custom time zones that adhere to the following time zone format:
  - General Time Zone: For time zones representing a GMT offset value, the following syntax is used:
    - Custom ID:
      - GMT Sign Hours: Minutes

- GMT Sign Hours Minutes
- GMT Sign Hours Hours

- Sign: one of:
  - + -

- Hours:
  - Digit
  - Digit Digit

- Minutes:
  - Digit Digit

- Digit: one of:
  - 0 1 2 3 4 5 6 7 8 9

NOTE: Hours are between 0 and 23, and minutes are between 00 and 59. For example, GMT +10 and GMT +0010 equates to ten hours and ten minutes ahead of GMT.

- When creating a TimeZone, the specified Custom Time Zone ID is normalized in the following syntax:
  - NormalizedCustomID:
    - GMT Sign TwoDigitHours: Minutes

    - Sign: one of:
      - + -

    - TwoDigitHours:
      - Digit Digit

    - Minutes:
      - Digit Digit

    - Digit: one of:
      - 0 1 2 3 4 5 6 7 8 9

MIDP 2.0 Specific Information for JTWI

MIDP 2.0 provides the library support for user interface, persistent storage, networking, security, and push functions. MIDP 2.0 contains a number of optional functions, some of which are implemented as outlined below. The JTWI requirements for MIDP 2.0 supports the following points:

- Record Store Minimum permits a MIDlet suite to create at least 5 independent RecordStores. This requirement does not intend to mandate that memory be reserved for these Record Stores, but it is possible to create the RecordStores if the required memory is available.
- HTTP Support for Media Content provides support for HTTP 1.1 for all supported media types. HTTP 1.1 conformance matches the MIDP 2.0 specification. See `javax.microedition.io` package for specific requirements.
- JPEG for Image Objects — ISO/IEC JPEG together with JFIF are supported. The support for ISO/IEC JPEG only applies to baseline DCT, non-differential, Huffman coding, as defined in JSR-185 JTWI specification, symbol 'SOF0'. This support extends to the `javax.microedition.lcdui.Image` class, including the methods outlined above. This mandate is voided in the event that the JPEG image format becomes encumbered with licensing requirements.
- Timer Resolution permits an application to specify the values for the `firstTime`, `delay`, and `period` parameters of `java.util.timer.schedule()` methods with a distinguishable resolution of no more than 40 ms. Various factors (such as garbage collection) affect the ability to achieve this requirement. At least 80% of test attempts will meet the schedule resolution requirement to achieve acceptable conformance.
- Minimum Number of Timers allows a MIDlet to create a minimum of 5 simultaneously running Timers. This requirement is independent of the minimum specified by the Minimum Application Thread Count.
- Bitmap Minimums support the loading of PNG images with pixel color depths of 1, 2, 4, 8, 16, 24, and 32 bits per pixel, per the PNG format specification. For each of these color depths, as well as for JFIF image formats, a compliant implementation supports images up to 76800 total pixels.
- TextField and TextBox and Phonebook Coupling — when the center select key is pressed while in a TextBox or TextField and the constraint of the TextBox or TextField is `TextField.PHONENUMBER`, the names in the Phonebook are displayed in the "Insert Phonenumber?" screen.
- Supported characters in TextField and TextBox — TextBox and TextField with input constraint `TextField.ANY` supports inputting all the characters listed in JSR-185.
- Supported characters in EMAILADDR and URL Fields — Class `javax.microedition.lcdui.TextBox` and

`javax.microedition.lcdui.TextField` with either of the constraints `TextField.EMAILADDR` or `TextField.URL` allows the same characters to be input as are allowed for input constraint `TextField.ANY`.
- Push Registry Alarm Events will implement alarm-based push registry entries.
- Identification of JTWI via system property — to identify a compliant device and the implemented version of this specification, the value of the system property `microedition.jtwi.version` is 1.0.

Wireless Messaging API 1.1 (JSR-120) Specific Content for JTWI

WMA defines an API used to send and receive short messages. The API provides access to network-specific short message services, such as GSM SMS or CDMA short messaging. JTWI supports the following as it is outlined in the <link type="internal" destination="JSR_120">JSR-120</link> chapter.

- Support for SMS in GSM devices
- Cell Broadcast Service in GSM devices
- SMS Push

Mobile Media API 1.1 (JSR-135) Specific Content for JTWI

The following are supported for JTWI compliance:

- HTTP 1.1 Protocol is supported for media file download for all supported media formats.
- MIDI feature set specified in MMAPI (JSR-135) is implemented. MIDI file playback is supported.
- VolumeControl is implemented and is required for controlling the volume of MIDI file playback.
- JPEG encoding in video snapshots is supported if the handset supports the video feature set and video image capture.
- Tone sequence file format is supported. Tone sequences provide an additional simple format for supporting the audio needs of many types of games and other applications.

21
JSR-205: WMA 2.0

Overview

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

Messaging Functionality

This section describes messaging functionality to be implemented by WMA.

MMS Message Structure

The MMS PDU structure is implemented as specified in the WAP-209-MMS Encapsulation standard. The MMS PDU consists of headers and a multipart message body. Some of the headers originate from standard RFC 822 headers and others are specific to multimedia messaging. In addition to defined MMS headers, it also contains header parameters as defined by JSR-205. The message body may contain parts of any content type. The MIME multipart is used to represent and encode a wide variety of media types for transmission via multimedia messaging.

MMS Message Addressing

The multipart message addressing model contains different types of addresses:

- global telephone number of recipient user, including telephone number, ipv4, ipv6 addresses
- e-mail address as specified in RFC 822
- short-code of the service (not valid for MMS version 1.0)

The syntax of the URL connection strings follow the rules specified in the JSR-205 specification.

MMS Message Types

MMS messages can be sent using MULTIPART_MESSAGE type of this API. The default type of message is multipart/related. If the content type header does not contain start parameter, the message type is assumed to be multipart/mixed. This section describes Multipart Message and its related classes. Messaging framework is described in the JSR-120 chapter.

MultipartMessage

The WMA implements the MultipartMessage an interface representing a multipart message. This is a subinterface of Message that contains methods to add, remove, and manipulate message parts. The interface also specifies the subject of the message.

Refer to the JSR-205 specification for more details.

MessagePart

The WMA implements the MessagePart class, representing a media part that can be sent with the message. Instances of MessagePart class are added to the MultipartMessage.

Each message part consists of part header and part body. The part headers include Content ID, Content Location, Content type, and Encoding scheme. Content can be of any MIME type.

Multimedia Message Service Center Address

The MMSC address used for sending the messages should be made available using `System.getProperty` with property name "`wireless.messaging.mms.mmsc`". Applications might need to obtain the Multimedia Message Service Center (MMSC) address to decide which recipient to use. For example, the application might need to do this because it is using service numbers for application servers that might not be consistent in all networks and MMSCs.

Refer to the JSR-205 specification for more details.

Application ID

The WMA supports sending of MMS messages to concrete Java applications.

Messages can be sent using this API via client or server type Message Connections. Refer to the JSR-205 specification for more details.

The application specifies Application ID when opening the server mode MessageConnection. The receiving application running on a device is identified with the Application ID included in the message.

The maximum number of Application IDs are limited by the implementation and depends on phone RAM availability and carrier operators preloaded content memory consumption.

The maximum number of simultaneously opened connections are limited by the implementation and depends on phone RAM availability and carrier operators preloaded content memory consumption.

The maximum number of MMS messages in the buffer at the same time are limited by the implementation and depends on phone RAM availability and carrier operators preloaded content memory consumption.

MMS Push

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

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

Requirements for WMA

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

MMS-push mechanism is implemented as described in the MIDP 2.0 chapter and some specific requirements are defined next.

Initial Setup

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

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

Handling the Incoming MMS Message

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

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

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

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

The WMA passes the received MMS messages to concrete Java applications associated with the Application ID.

Application is Running/Resuming

The application startup and resume is implemented in accordance with requirements outlined in the MIDP 2.0 chapter.

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

Application is Running/Background

The application receiving the incoming MMS message handles this MMS message.

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

An application is responsible to handle a corresponded (Application ID) received

message (store it more persistently if needed). An MMS message may get lost if an application can not save it due to lack of space.

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

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

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

Application is Suspending

The application suspending is implemented in accordance with requirements outlined in the MIDP 2.0 chapter.

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

Application is Ending

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

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

MMS Push

When receiving a message that has an unknown Application ID, the MMS Engine validates the routing options registered by each of its clients.

Since the Application ID does not match with the routing parameters, a NotifyResponse is sent back to the MMSC, with status set to REJECTED.

When receiving a message that has an unknown Application ID / from_address, MMS Engine validates the routing options registered by each of its clients.

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

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

The transient notice has the following UI dialog elements (to be used for P04.4 integration releases):

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

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

The decisions to deploy these options are controlled by a Feature-ID as it depends on the behavior desired by the Operator.

Requirements to the Native MMS Client

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

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

Anonymous Rejection Feature

The Native MMS Client supports anonymous rejection feature.

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

Filtering of the `from_address` is done at the Notification-Level.

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

Coincidental Addresses in the Native Client and Java Clients Address Filters

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

Messages received with these addresses are rejected.

The Java client maintains an Acceptable list of address-filters: only Messages that match this Address-filter are handled by the Java client.

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

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

Security Policy

The WMA follows the security policy specified in the MIDP 2.0 chapter.

To send and receive messages using WMA, applications are granted permission to perform the requested operation. The following table assigns individual permissions:

Permission	Pro-tocol	Function
`javax.microedition.io.Connector.mms`	mms	`Connector.open("mms://....")`

`javax.wireless.messaging.mms.receive`	mms	`MessageConnection.setMessageListener` `MessageConnection.receive`
`javax.wireless.messaging.mms.send`	mms	`MessageConnection.send`

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

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

VMVM Support

WMA functionality is supported in VMVM environment.

External Event Interaction

The implementation follows external event interactions.

22 Motorola Get URL from Flex API

Overview

The existing functionality allows current Java Applications to use a dedicated URL to inform users about the location from which a new level of game can be downloaded. This new functionality allows carriers to specify the URL for content download. This feature allows accessing URLs stored in FLEX by a Java application. Carriers flex the URL, which is used for content download, into the phone just like any invisible net URL.

Flexible URL for Downloading Functionality

The URL is flexed using RadioComm or using OTA provisioning. The following rules apply:

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

This feature enables Java applications to read the URL stored at the predefined location in the flex table.

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

Security Policy

Only trusted applications are granted permission to access this property.

Appendix A
Key Mapping

Key Mapping

Table 33 identifies key names and corresponding Java assignments. All other keys are not processed by Java.

Key	Assignment
JOYSTICK LEFT	LEFT
JOYSTICK RIGHT	RIGHT
JOYSTICK UP	UP
JOYSTICK DOWN	DOWN
SCROLL UP	UP
SCROLL DOWN	DOWN
SOFTKEY 1	SOFT1
SOFTKEY 2	SOFT2
SEND	SELECT Also, call placed if pressed on lcdui.TextField or lcdui.TextBox with PHONENUMBER constraint set.
CENTER SELECT	SELECT
END	Handled according to Motorola specification: Pause/End/Resume/Background menu invoked.

Table 33 Key Mapping

Appendix B Memory Management Calculation

Available Memory

The available memory on the MOTOROKR E6/E6e is the following:

- 16MB shared memory for MIDlet storage
- 2Mb Heap size

Appendix C
FAQ

Online FAQ

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

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

http://developer.motorola.com

Appendix D
HTTP Range

Graphic Description

Figure 17 shows a graphic description of HTTP Range:

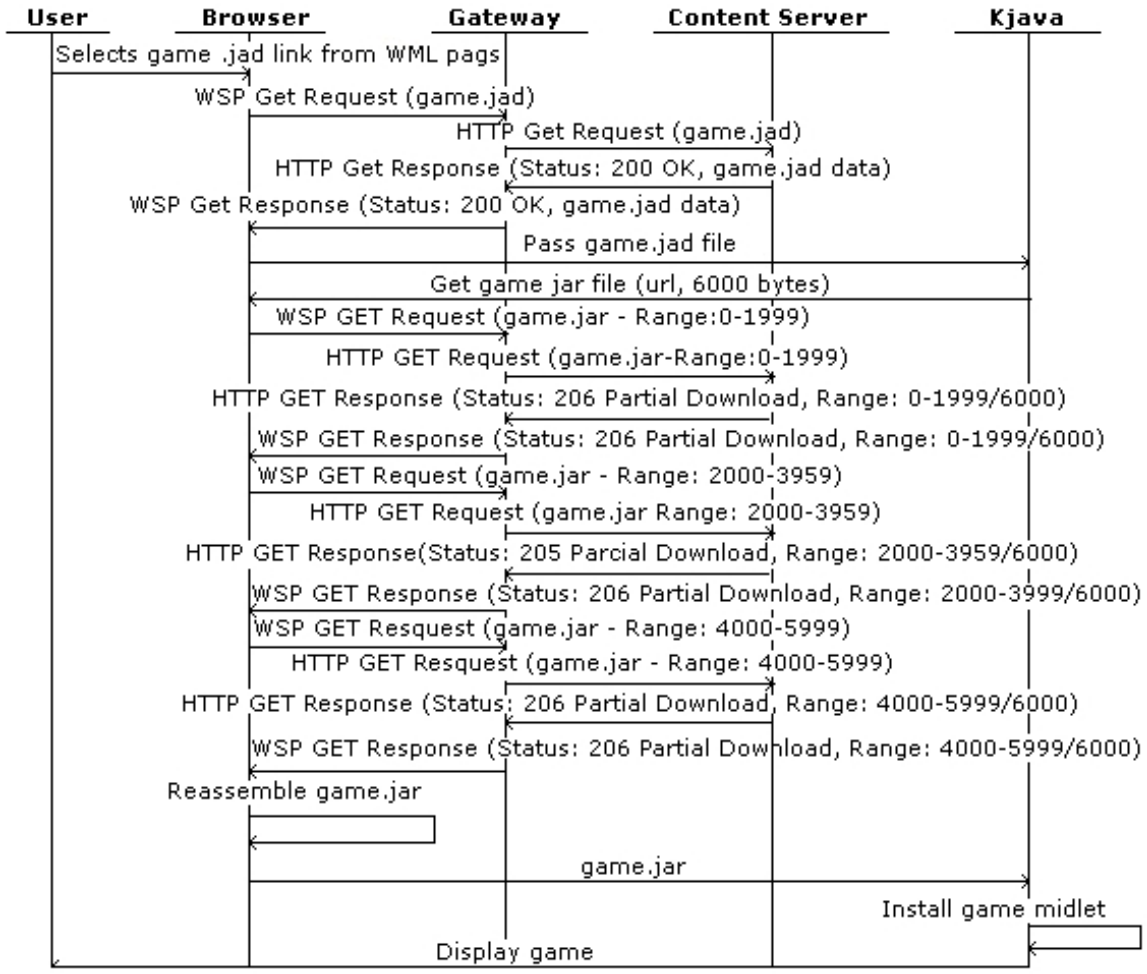


Figure 17 Graphic Description of HTTP Range