Technical Manual

Motorola C381p Handset J2ME™ Developer Guide

Version 01.00

Table of Contents

Table of Figures

Index of Tables

Table of Code Samples

1 Introduction

Purpose

This document describes the application program interfaces used to develop Motorola compliant Java™ 2 Platform, Micro Edition (J2ME™) applications for the C381p handset.

Audience

This document is intended for premium J2ME developers and specific carriers involved with the development of J2ME applications for the C381p handset.

Disclaimer

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

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

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

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

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting form use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or

in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

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

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

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

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

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

References

Reference	Link
RFC 2068	http://ietf.org/rfc/rfc2068.txt
SAR	http://www.wapforum.org
MIDP 2.0	http://java.sun.com/products/midp/
JSR 118	http://www.jcp.org
JSR 120	http://www.jcp.org
JSR 135	http://www.jcp.org
Sun MIDP 2.0 SDK	http://java.sun.com/products/midp/
TLS protocol version 1.0	http://www.ietf.org/rfc/rfc2246.txt
SSL protocol version 3.0	http://home.netscape.com/eng/ssl3/draft302.txt
GSM 03.38 standard	http://www.etsi.org
GSM 03.40 standard	http://www.etsi.org
RFC 2437	http://ietf.org/rfc/rfc2437.txt
Sun J2ME	http://java.sun.com/j2me/).

Revision History

Version	Date	Name	Reason
00.01	November 09, 2004	C.E.S.A.R.	Initial Draft

Definitions, Abbreviations, Acronyms

Acronym	Description
AMS	Application Management Software
API	Application Program Interface.
CLDC	Connected Limited Device Configuration
GPS	Global Positioning System
IDE	Integrated Development Environment
ITU	International Telecommunication Union
JAD	Java Application Descriptor
JAL	Java Application Loader
JAR	Java Archive. Used by J2ME applications for compression and packaging.
J2ME	Java 2 Micro Edition
JSR 120	Java Specification Request 120 defines a set of optional APIs that provides standard access to wireless communication resources.
JVM	Java Virtual Machine
KVM	Kilo Virtual Machine
LWT	Lightweight Windowing Toolkit
MIDP	Mobile Information Device Profile
MMA	Multimedia API
MT	Mobile Terminated
OEM	Original Equipment Manufacturer
OTA	Over The Air
RMS	Record Management System

RTOS	Real Time Operating System
SDK	Software Development Kit
SMS	Short Message Service
SMSC	Short Messaging Service Center
SU	Subscribe Unit
UI	User Interface
URI	Unified Resource Identifier
VM	Virtual Machine
WMA	Wireless Messaging API

Document Overview

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

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

Chapter 2 – J2ME Introduction: this chapter describes the J2ME platform and the available resources on the Motorola C381p handset.

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

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

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

Chapter 6 – JAD Attributes: this chapter describes what attributes are supported.

Chapter 7 – Java.lang Implementation: this chapter describes the java.lang implementation.

Chapter 8 – Networking APIs: this chapter describes the Java Networking API.

Chapter 9 – JSR 135 Mobile Media: this chapter describes image types and supported formats.

Chapter 10 – JSR 120 Wireless Messaging API: this chapter describes JSR 120 implementation.

Chapter 11 -- Phonebook Access API: this chapter describes the Phonebook Access API.

Chapter 12 – Telephony API: this chapter describes the Telephony API.

Chapter 13 – Serial Port Access: this chapter describes the Serial Port Access.

Chapter 14 – SMS Messaging as GSM Extension: this chapter describes the SMS Access API.

Chapter 15 – User Display Interface: this chapter describes the J2ME specific Canvas, Hardware Mapping and External Event Interaction functionality.

Chapter 16 – One-Click Application Access: this chapter describes the used Java applications via the soft keys, navigation keys or smart keys

Chapter 17 – Download MIDlet Through Browser: this chapter describes the performing any downloads on the handset.

Chapter 18 – Lightweight Windowing Toolkit: this chapter describes the capabilities to include a component-level API through which developers can control the co intents and layout of their screens.

Chapter 19 – UDP Support: this chapter describes how to enable J2ME applications access to Generic UDP Transport Service.

Chapter 20 – Shared JAD URLS: this chapter describes briefly a new feature that allows users to share their downloaded J2ME application URLs with others.

Chapter 21 – Get URL from Flex API: this chapter describes the way to access URL stored in FLEX by a java application.

Chapter 22 – File System Access API: this chapter describes the File System API.

Chapter 23 – Multiple Key Press: this chapter describes the Multiple Key Press.

Chapter 24 – ITAP: this chapter describes iTAP support.

Chapter 25 – LCDUI: this chapter describes the LCDUI.

Chapter 26 – Auto Launch of Midlet: this chapter describes the Auto Lanch of Midlet.

Chapter 27 – Background Applications: this chapter describes the.

Chapter 28 – Java System Menu: this chapter describes the Java System Menu.

Chapter 29 – Invisible net for J2ME: this chapter describes the Invisible net for J2ME.

Chapter 30 – Download MIDlet Through PC: this chapter describes the any downloads on the handset.

Chapter 31 – MIDP 2.0 Security Model: this chapter describes the MIDP 2.0 default security model.

Appendix A – Key Mapping: this appendix describes the key mapping of the Motorola C381p handset, including the key name, key code, and game action of all Motorola keys.

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

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

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

Appendix E – Spec Sheet: this appendix provides the spec sheet for the Motorola C381p handset.

2 J2ME Introduction

The Motorola C381p handset includes the Java™ 2 Platform, Micro Edition, also known as the J2ME platform. The J2ME platform enables developers to easily create a variety of Java applications ranging from business applications to games. Prior to its inclusion, services or applications residing on small consumer devices like cell phones could not be upgraded or added to without significant effort. By implementing the J2ME platform on devices like the Motorola C381p handset, service providers, as well as customers, can easily add and remove applications allowing for quick and easy personalization of each device. This chapter of the guide presents a quick overview of the J2ME environment and the tools that can be used to develop applications for the Motorola C381p handset.

The Java 2 Platform, Micro Edition (J2ME)

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

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

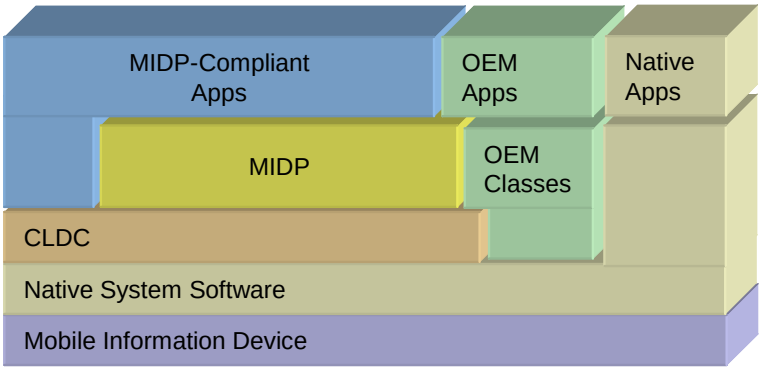


Figure 1 Java Platform

The Configuration Layer used in the Motorola C381p handset is the Connected Limited Device Configuration 1.1 (CLDC 1.1) and the Profile Layer used is the Mobile Information Device Profile 2.0 (MIDP 2.0). Together, the CLDC and MIDP provide common APIs for I/O, simple math functionality, UI, and more.

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

The Motorola J2ME Platform

Functionality not covered by the CLDC and MIDP APIs is left for individual OEMs to implement and support. By adding to the standard APIs, manufacturers can allow developers to access and take advantage of the unique functionality of their handsets. The Motorola C381p handset contains OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the Motorola C381p 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.

MIDP 1.0

J2ME is the version of Java that the mobile device will support. It was developed to support devices with limited memory, i.e mobile devices, pagers, SIM cards.
J2ME maintains the qualities that Java technology has become famous for:

- built-in consistency across products in terms of running anywhere, any time, over any device
- portability of the code
- leveraging of the same Java programming language
- safe network delivery
- applications written with J2ME are upwardly scalable to work with J2SE and J2EE.

J2ME enables device manufacturers, service providers, and content creators to deploy compelling new applications and services to their customers rapidly and cost-effectively while capitalizing on new revenue streams.
In using J2ME, the handset must be MIDP 1.0 and CLDC 1.0 compliant. To assure this compliance, the handset must pass the Technology Certification Kit, TCK, provided by Sun.

Resources and API's Available

MIDP 2.0 will provide support to the following functional areas on the Motorola C381p handset:

MIDP 2.0

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

Additional Functionality

- WMA (JSR 120)
- MMA (JSR 135)
- Phonebook API
- Telephony API

3
Developing and Packaging J2ME Applications

Guide to Development in J2ME

Introduction to Development

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

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

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

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

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

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

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

For beginning development, key points to remember are memory size, processing power, screen capabilities and wireless network characteristics. These all play an important part

in development of a MIDlet. The specifications listed above are designed to work upon devices that have these characteristics.

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

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

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

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

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

4 Downloading Applications

Method of Downloading

The option open to the developer for deploying the MIDlet to a physical Motorola device is OTA (over -the-air) downloading.

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 OPERA 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 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 100k in size.

During a download of the application, the user will see the OPERA browser displaying a progress dialog.

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

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
- 913 Required package not supported by device
- 999 Other errors

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 will display a dialog stating "Installation failed. Package invalid." To dismiss this dialog, press "OK".
- When downloading is done, the handset displays a transient notice "Download Completed" and starts to install the application.
- Upon completing installation, the handset displays a dialog "Install complete". To dismiss this dialog, press "OK".
- If the MANIFEST file is wrong, the handset displays a dialog stating "Installation failed. Package invalid." To dismiss this dialog, press "OK".
- If JAD does not contain mandatory attributes, "Installation failed. Package invalid." notice appears.

Error Logs

The Table 1 represents the error logs associated with downloading applications.

Error Logs	Scenario	Possible Cause	Error Dialog
906 Invalid Descriptor.	JAD Download	Missing or incorrectly formatted mandatory JAD attributes Mandatory: MIDlet-Name (up to 32 symbols) MIDlet-Version MIDlet-Vendor (up to 32 symbols) MIDlet-JAR-URL (up to 256 symbols) MIDlet-JAR_Size JAD signature verification failed Unknown error during JAD validation	Failed: Invalid File
904 JAR Size	OTA JAR	The received JAR size does not	Download Failed

Mismatch .	Download	match the size indicated in JAD	
902 User Cancelled.	OTA JAR Download	User cancelled download	Cancelled: <Icon> <Filename>
903 Loss of Service.	OTA JAR Download	Browser lost connection with server	Installation Failed
901 Insufficient Memory.	OTA JAR Download	Insufficient space to install the MIDlet suite	Insufficient Storage
905 Attribute Mismatch	Installation	Mandatory attributes are not identical in JAD & Manifest	Installation failed. Package invalid.
901 Insufficient Memory.	Installation	Insufficient Space to install MIDlet suite	Insufficient Storage
907 Invalid JAR.	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	Installation failed. Package invalid.
	MIDlet Launching	Security Certificates expired or removed	Application Expired
	MIDlet Execution	Authorization failure during MIDlet execution Incorrect MIDlet	Application Error

Table 1 Error Logs

OTA and Download

- Comply with “OTA User Initiated Provisioning Specifications” in MIDP 2.0.
- The user MUST be prompted if the midlet is chargeable.
- Terminals should compare and predict the application program that will be downloaded is
- the latest or old versions and give indication to users.
- Check the available memory before any downloads.
- Users should be able to terminate the download process any time by pressing END key.
- Before downloading the following JAD file information must be displayed first:

Contents	Maximum length
Application program name (Midlet name)	32 Bytes Max
Application program version number	16 Bytes Max
Vendor Name	32 Bytes Max

URL of JAR file	Not specified
Size of JAR file	8 Bytes Max
Applicable Terminal Type	not specified
Application program introduction (Midlet description)	512 Bytes Max
Information Fee (Media price).	32 Bytes Max

Table 2 JAD file information

- End user should be able to delete the applications downloaded.

5 Application Management

The following sections describe the application management scheme for the Motorola C381p handset. This chapter will discuss the following:

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

Downloading a JAR file without a JAD

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

MIDlet Upgrade

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

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

- If the cryptographic signer of the new MIDlet suite and the original MIDlet suite are identical, then the RMS record stores MUST be retained and made available to the new MIDlet suite.
- If the URL of the new MIDlet suite is identical to the URL the original MIDlet suite was downloaded from, then the RMS MUST be retained and made available to the new MIDlet suite.
- If the above statements are false, then the device MUST ask the user whether the data from the original MIDlet suite should be retained and made available to the new MIDlet suite.

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

Installation and Deletion Status Reports

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

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

Feature/Class
Application upgrades performed directly through the AMS
When removing a MIDlet suite, the user will be prompted to confirm the entire MIDlet suite will be removed
Prompt for user approval when the user has chosen to download an application that is identical to, or an different version of an application currently in the handset
Unauthorized MIDlets will not have access to any restricted function call
AMS will check the JAD for security indicated every time a installation is initiated
Application descriptor or MIDlet fails the security check, the AMS will prevent the installation of that application and notify the user that the MIDlet could not be installed
Application descriptor and MIDlet pass the security check , the AMS will install the MIDlet and grant it the permissions specified in the JAD
A method for launching Java application that maintains the same look and feel as other features on the device will be provided
User will be informed of download and installation with progress indicator and will be given an opportunity to cancel the process
User will be prompted to launch the MIDlet after installation
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/class support for MIDP 2.0

System Menu

The Java System Menu can be found under the Java Settings menu under the Main Menu. The Java System menu allows the user to see what version of MIDP and CLDC is being used in the handset. It also shows the user the free data space available, program space available, and the heap size being used. The following table describes each function in detail.

Refer to the Table 4 Java System menu:

Action	Description
CLDC Version	This displays the CLDC version that is being used in the handset.
MIDP Version	This displays the MIDP version that is being used in the handset.
Data Space	This displays the amount of free memory available for data used by the applications, i.e. phone book entries, game high scores..
Program Space	This displays the amount of free memory available for applications.
Heap Size	This is the amount of runtime memory available in the handset for J2ME applications.

Table 4 Java System menu

6
JAD Attributes

JAD / Manifest Attribute Implementations

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

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

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

Motorola has implemented the following support for the MIDP 2.0 Java Application Descriptor attributes as outlined in the JSR-118. The Table 5 ists all MIDlet attributes, descriptions, and its location in the JAD and/or JAR manifest that are supported in the Motorola implementation.

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.		
MIDlet-Description	The description of the MIDlet suite.		

MIDlet-Info-URL	A URL for information further describing the MIDlet suite.		
MIDlet-<n>	The name, icon, and class of the nth MIDlet in the JAR file. Name is used to identify this MIDlet to the user. Icon is as stated above. Class is the name of the class extending the javax.microedition.midlet.MIDletclass.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MIDlet-Jar-URL	The URL from which the JAR file can be loaded.		Yes
MIDlet-Jar-Size	The number of bytes in the JAR file.		Yes
MIDlet-Data-Size	The minimum number of bytes of persistent data required by the MIDlet.		
MicroEdition-Profile	The J2ME profiles required. If any of the profiles are not implemented the installation will fail.	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MicroEdition-Configuration	The J2ME Configuration required, i.e CLDC 1.0	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MIDlet-Permissions	Zero or more permissions that are critical to the function of the MIDlet suite.		
MIDlet-Permissions-Opt	Zero or more permissions that are non-critical to the function of the MIDlet suite.		
MIDlet-Push-<n>	Register a MIDlet to handle inbound connections		
MIDlet-Install-Notify	The URL to which a POST request is sent to report installation status of the MIDlet suite.		
MIDlet-Delete-Notify	The URL to which a POST request is sent to report deletion of the MIDlet suite.		
MIDlet-Delete-Confirm	A text message to be provided to the user when prompted to confirm deletion of the MIDlet suite.		
Background	MIDlets with this Motorola specific attribute will continue to run when not in focus.		

Table 5 MIDlet Attributes, descriptions, and JAD and/or JAR location

7 Java.lang Implementation

java.lang support

Motorola implementation for the java.lang.System.getProperty method will support additional system properties beyond what is outlined in the JSR 118 specification and is controlled by a flex bit. These additional system properties can only be accessed by trusted MIDlets.

The additional system properties are as follows:

- Cell ID: The current Cell ID of the device will be returned during implementation.
- IMEI: The IMEI number of the device will be returned during implementation.

The Code Sample 1 shows java.lang support:

```
System.getProperty("phone.mcc")
System.getProperty("phone.mnc")
System.getProperty("phone.imei")
System.getProperty("phone.cid")
System.getProperty(“phone.lai”)
System.getProperty(“phone.ta”)
```

Code Sample 1 Java.lang support

8 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
- Socket connection
- SSL

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

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

SecureConnection interface in the javax.microedition.io.package	Supported
SecurityInfo Interface in the javax.microedition.io.package	Supported
ServerSocketConnection interface in the javax.microedition.io.package	Supported
UDPDatagramConnection interface in the javax.microedition.io.package	Supported

Table 6 Network API feature/class support for MIDP 2.0

The Code Sample 2 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 2 Socket Connection

User Permission

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

HTTPS Connection

Motorola implementation supports a HTTPS connection on the Motorola C381p handset. Additional protocols that will be supported are the following:

- TLS protocol version 1.0 as defined in http://www.ietf.org/rfc/rfc2246.txt
- SSL protocol version 3.0 as defined in http://home.netscape.com/eng/ssl3/draft302.txt

The Code Sample 3 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 3 HTTPS Connection

9
JSR 135 Mobile Media API

JSR 135 Mobile Media API

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

- Tone Sequence
- Sampled Audio
- MIDI

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

The Code Sample 4 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 =
Manager.createPlayer(getClass().getResourceAsStream("MP3.mp3
"), "audio/mpeg");
}
catch (Exception e)
{
    System.out.println("FAILED: exception for createPlayer:
" + e.toString());
}

// Obtain the information required to acquire the media
resources
try
{
    player.realize();
}
```

```
            catch (MediaException e)
            {
                System.out.println("FAILED: exception for realize: " +
            e.toString());
            }

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

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

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

            // Release the resources
            player.close();
```

Code Sample 4 JSR 135 Mobile Media API

ToneControl

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

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

The following is the available method for ToneControl:

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

GUIControl

GUIControl extends control and is defined for controls that provide GUI functionalities. USE_GUI_PRIMITIVE defines a mode on how the GUI is displayed. initializes the mode on how the GUI is displayed.

When USE_GUI_PRIMITIVE is specified for initDisplayMode, a GUI primitive will be returned. This object is where the GUI of this control will be displayed. It can be used in conjunction with other GUI objects and conforms to the GUI behaviors specified.

VolumeControl

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

The following describes the different volume settings found within VolumeControl:

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

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

`-getLevel`: Get the current volume setting.

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

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

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

StopTimeControl

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

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

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

Manager Class

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

Audio Media

The Table 7 depicts the multimedia file formats are supported:

File Type	CODEC
WAV	PCM
SP MIDI	General MIDI
MIDI Type 1	General MIDI
MP3	MPEG-1 layer III
MIDI Type 0	General MIDI

Table 7 Multimedia File formats

The Table 8 depicts a list of audio MIME types supported:

Category	Description	MIME Type
Audio/Video/Image	Wav	`Audio/x-wav`
	MP3 Audio	`Audio/mpeg`
	MIDI Audio	`Audio/midi`
	SP MIDI Audio	`Audio/sp-midi`

	Tone Sequences Audio	`Audio/x-tone-sequence`

Table 8 List of audio MIME types

Refer to the Table 9 for multimedia feature/class support for JSR 135:

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

Table 9 Multimedia feature/class support for JSR 135

Note: If two wave plays have the same frequency, they can mix. See Appendix A – mix audio table.

Player number limitation <=4 and MIDI size limitation is 150K

MP3 stream size has the limitation

10 JSR 120 – Wireless Messaging API

Wireless Messaging API (WMA)

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

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

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

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

- Creating a SMS
- Sending a SMS
- Receiving a SMS
- Viewing a SMS
- Listening to a SMS

SMS Client Mode and Server Mode Connection

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

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

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

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

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

SMS Port Numbers

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

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

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

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

JSR 120 Section A.6.0 Restricted Ports:
2805, 2923, 2948, 2949, 5502, 5503, 5508, 5511, 5512, 9200, 9201, 9202, 9203, 9207, 49996, 49999.

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

SMS Message Types

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

SMS Message Structure

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

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

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

SMS Notification

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

- A MIDlet will handle an incoming SMS message if the MIDlet is registered to receive messages on the port (identifier) and is running.
- When a MIDlet is paused and is registered to receive messages on the port number of the incoming message, then the user will be queried to launch the MIDlet.
- If the MIDlet is not running and the Java Virtual Machine is not initialized, then a Push Registry will be used to initialize the Virtual Machine and launch the J2ME MIDlet. This only applies to trusted, signed MIDlets.
- If a message is received and the untrusted unsigned application and the KVM are not running then the message will be discarded.

The Table 10 depicts list of Messaging features/classes supported in the device.

Feature/Class	Implementation
JSR-120 API. Specifically, APIs defined in the javax.wireless.messaging package will be implemented with regards to the GSM SMS Adaptor	Supported
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	5
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
Number of concatenated messages.	40 messages in inbox, each can be concatenated from 10 parts at max. No limitation on outbox (immediately transmitted)

Table 10 Messaging features/classes supported

The Code Sample 5 shows the implementation of the JSR 120 Wireless Messaging API:

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

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

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

   /* Create BinaryMessage for client mode */
   binMsg =
(BinaryMessage)connClient.newMessage(MessageConnection.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 timestamp for the message:

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

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

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

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

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

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 Wireless Messaging API

11 Phonebook Access API

Phonebook Access API

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

The primary goal of the Phonebook Access API is to be simple and thin to fit in resource-limited devices like the Motorola C381p handset. This API will specify a base storage class for all types of contacts items presented in the vCard specification (RFC2426 – vCard MIME Directory Profile – vCard 3.0 Specification). In addition, schema strings used in querying and storing contact information are those specified in the RFC2426 specification.

The Phonebook Access API will perform the following functions:

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

Phonebook Access API Permissions

Prior to a MIDlet accessing the Phonebook API for all Phonebook operations, the implementation will check the Phonebook permissions granted to the application.

In the Motorola C381p, Phonebook API permissions have been added to the MIDP 2.0 security framework "com.motorola.phonebook." The behavior is up to the domain setting where the MIDlet falls in. For an untrusted MIDlet, the permission for the API is "Oneshot".

The Code Sample 6 shows the implementation of the Phonebook API:

Sample of code to create object of PhoneBookRecord class:

```
PhoneBookRecord phbkRecEmpty = new PhoneBookRecord();

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

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

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

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

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

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

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

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

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

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

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

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

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

```
index = phbkRec.findRecordByTelNo(telNo,
PhoneBookRecord.SORT_BY_NAME);
```

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

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

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

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

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

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

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

```
// get number of available records
int numberRecsAvalPhone =
PhoneBookRecord.getAvailableRecords(PhoneBookRecord.PHONE_MEMORY);
int numberRecsAvalSim =
   PhoneBookRecord.getAvailableRecords(PhoneBookRecord.SIM_MEMORY);
int numberRecsAvalAll =
PhoneBookRecord.getAvailableRecords(PhoneBookRecord.ALL_MEMORY);
```

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

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

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

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

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

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

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

```
int maxTelNoLengthPhone =
PhoneBookRecord.getMaxTelNoLength(PhoneBookRecord.PHONE_MEMORY);
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Code Sample 6 Phonebook API

12 Telephony API

The Telephony API allows a MIDlet to make a voice call, however, the user needs to confirm the action before any voice call is made. The reason for the confirmation is to provide a measure of security from rogue applications overtaking the handset.

Unlike standard TAPI, the wireless Telephony API provide simple function and simple even listener: startCall (), send ExtNo (), and endCall (), DialerListener.

The Tables 11 and 12 describe the Interface and Class Summary:

Interface Summary	
DialerListener	The DialerListener interface provides a mechanism for the application to be notified of phone call event.

Table 11 Interface Summary

Class Summary	
Dialer	The Dialer class defines the basic functionality for start and end phone call.
DialerEvent	The DialerEvent class defines phone call events.

Table 12 Class Summary

Dialer Class

The dialer Class can be used to start and end a phone call and user listener to receive an event. The applications use a Dialer to make a phone call and use DialerListener to receive event.

Class DialerEvent

The DialerEvent class defines phone call events. The Table 13 describes the Field Summary:

Summary	
static byte	PHONE_VOICECALL_CONNECT Phone call was connected event
static byte	PHONE_VOICECALL_DISCONNECT Phone call was disconnected event
static byte	PHONE_VOICECALL_DTMF_FAILURE Send extension number error event
static byte	PHONE_VOICECALL_FAILURE start phone call error event
static byte	PHONE_VOICECALL_HOLD Current java phone call was held by native phone event
static byte	PHONE_VOICECALL_INIT Phone start dial-up event
static byte	PHONE_VOICECALL_TIMEOUT Phone process timeout event
static byte	PHONE_VOICECALL_UNHOLD Current java phone call was unheld event

Table 13 Field Summary

The Table 14 describes the Constructor Summary:

Constructor Summary
DialerEvent()

Table 14 Constructor Summary

The following methods are inherited from class java.lang.Object:

- equals
- getClass
- hashCode
- notify
- notifyAll
- toString
- wait

The Table 15 describes the Field Details:

Field	Detail	Definition
PHONE_VOICECALL_INIT	public static final byte PHONE_VOICECALL_INIT	Phone start dial-up event
PHONE_VOICECALL_FAILURE	public static final byte PHONE_VOICECALL_FAILURE	Start phone call error event

PHONE_VOICECALL_CONNECT	public static final byte PHONE_VOICECALL_CONNECT	Phone call was connected event
PHONE_VOICECALL_DISCONNECT	public static final byte PHONE_VOICECALL_DISCONNECT	Phone call was disconnected event
PHONE_VOICECALL_TIMEOUT	public static final byte PHONE_VOICECALL_TIMEOUT	Phone process timeout event
PHONE_VOICECALL_HOLD	public static final byte PHONE_VOICECALL_HOLD	Current java phone call was held by native phone event
PHONE_VOICECALL_UNHOLD	public static final byte PHONE_VOICECALL_UNHOLD	Current java phone call was unheld event
PHONE_VOICECALL_DTMF_FAILURE	public static final byte PHONE_VOICECALL_DTMF_FAILURE	Send extension number error event

Table 15 Field Details

Class Dialer

The Dialer class defines the basic functionality for starting and ending a phone call. The Table 16 describes the Method Summary:

Method Summary	
void	**endCall**() end or cancel a phone call
static Dialer	**getDefaultDialer**()

void	**sendExtNo**(String extNumber) send extension number.
void	**setDialerListener**(DialerListener listener) Registers a DialerListener object.
void	**startCall**(String telenumber) start a phone call using given telephone number.
void	**startCall**(String teleNumber, String extNo) start a phone call using given telephone number and extension number.

Table 16 Method Summary

The following methods are inherited from java.lang.object:

- equals
- getClass
- hashCode
- notify
- notifyAll
- toString
- wait

getDefaultDialer

```
public static Dialer getDefaultDialer()
```

Get a Dialer instance.

setDialerListener

public void setDialerListener(DialerListener listener)

Registers a DialerListener object.

The platform will notify this listener object when a phone event has been received to this

Dialer object.

There can be at most one listener object registered for a Dialer object at any given point in time. Setting a new listener will implicitly de-register the possibly previously set listener.

Passing null as the parameter de-registers the currently registered listener, if any.

Parameters:

listener - DialerListener object to be registered. If null, the possibly currently registered listener will be de-registered and will not receive phone call event.

startCall

public void **startCall**(`String` telenumber)
throws `IOException`

start a phone call using given telephone number.

Parameters:

`telenumber` - the telephone number to be call.

`extNo` - the extension number to be send.

Throws:

`IOException` - if the call could not be created or because of network failure

`NullPointerException` - if the parameter is null

`SecurityException` - if the application does not have permission to start the call

startCall

public void **startCall**(`String` teleNumber,
`String` extNo)
throws `IOException`

start a phone call using given telephone number and extension number.

Parameters:

`telenumber` - the telephone number to be call.
`extNo` - the extension number to be send.

Throws:

`IOException` - if the call could not be created or because of network failure
`NullPointerException` - if the parameter is null
`SecurityException` - if the application does not have permission to start the call

sendExtNo

public void **sendExtNo**(`String` extNumber)
throws `IOException`

send extension number.

Parameters:

`sendExtNo` - the extension number to be send.

Throws:

`IOException` - if the extension could not be send or because of network failure

endCall

public void **endCall**()
throws `IOException`
end or cancel a phone call
Throws:

`IOException` - if the call could not be end or cancel.

Interface DialerListener

public interface **DialerListener**
The `DialerListener` interface provides a mechanism for the application to be notified of phone call event.

When an event arrives, the `notifyDialerEvent()` method is called

The listener mechanism allows applications to receive TAPI voice call event without needing to have a listener thread

If multiple event arrive very closely together in time, the implementation has calling this listener in serial.

Sample DialerListener Implementation

Dialer listener program

```
import java.io.IOException;
 import javax.microedition.midlet.*;
 import javax.microedition.io.*;
 import com.motorola.*;

 public class Example extends MIDlet implements DialerListener {
     Dialer dialer;

     // Initial tests setup and execution.

     public void startApp() {
       try {
         dialer = Dialer.getDefaultDialer();

         // Register a listener for inbound TAPI voice call events.
         dialer.setDialerListener(this);
         dialer.startCall("01065642288");

       } catch (IOException e) {
         // Handle startup errors
       }
```

```
  }
```

Asynchronous callback for receive phone call event

```
    public void notifyDialerEvent(Dialer dialer, byte event) {
      switch (event) {
        case DialerEvent.PHONE_VOICECALL_INIT:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_FAILURE:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_CONNECT:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_DISCONNECT:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_TIMEOUT:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_HOLD:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_UNHOLD:
          // your process
            break;
        case DialerEvent.PHONE_VOICECALL_DTMF_FAILURE:
          // your process
            break;
      }
    }

    // Required MIDlet method - release the connection and
    // signal the reader thread to terminate.

    public void pauseApp() {
      try {
        dialer.endCall();
      } catch (IOException e) {
        // Handle errors
      }
    }

    // Required MIDlet method - shutdown.
    // @param unconditional forced shutdown flag

    public void destroyApp(boolean unconditional) {
      try {
        dialer.setDialerListener(null);
        dialer.endCall();
      } catch (IOException e) {
        // Handle shutdown errors.
      }
    }
}
```

Code Sample 7 DialerListener Implementation

notifyDialerEvent

public void notifyDialerEvent(`Dialer` dialer,

byte event)

Called by the platform when a phone call event was received by a Dialer object where the application has registered this listener object.

This method is called once for each TAPI voice call event to the Dialer object.

NOTE: The implementation of this method MUST return quickly and MUST NOT perform any extensive operations. The application SHOULD NOT receive and handle the message during this method call. Instead, it should act only as a trigger to start the activity in the application's own thread.

Parameters:

`dialer` - the Dialer where the TAPI voice call event has arrived

`event` - the TAPI voice call event type.

Class Hierarchy

- class java.lang.**`Object`**
  - class com.motorola.phone.**`Dialer`**
  - class com.motorola.phone.**`DialerEvent`**

Interface Hierarchy

- interface com.motorola.phone.**`DialerListener`**

13
Serial Port Access

The Serial Port RS232 API is incorporated into any J2ME device allowing connection to

the J2ME serial port communication stream.

Opening the port requires the form Connector.open("comm:/name;parameters) where name is the name of the serial port. (A comma separated list of supported names can be obtained by calling System.getProperty("serialport.name");)

The API should allow the J2ME application to query for the available device connections (data cable), as well as the available data speeds.

This API should be in-line with the CLDC javax.microedition.io Version 1.02. In addition, the following changes must be added over the CLDC standard:

- Automatic baud rate detection.
- Addition of more parity options; specifically mark and space.
- Support for 1.5 stop bits as a common UART feature.
- Changes to port names, as opposed to the "open by port number option". The use of actual port names are more meaningful to the user.
- The mode parameter for Connector.open() is not ignored. A MIDlet should not be allowed to write a connection which is specified as READ only.

14 SMS Messaging as GSM Extension

The SMS APIs for J2ME environment enable the following functionality:

- Creating an SMS message
- Sending an SMS message
- Viewing an SMS message
- Deleting an SMS message

The SMS APIs support the following contents to be sent over SMS:

- Text message
- Audio files: .au, .midi. .wav, .mp3
- Image files: .png, .gif

The SMS messages for each MIDlet are managed by the MIDlet.

Creating a Message

The API enables creating of SMS messages. The addressing is compliant with the standard SMS addressing as specified in the GSM standards. The content of the message can be constructed using one or more of the content types specified in chapter 10 of this document.

Sending a Message

The SMS API provides a way to send a message to the appropriate SMS address. Any errors while sending the message should be handled appropriately.

Viewing a Message

The APIs should provide a way to get a list of messages that is associated with a specific application and to view the messages.

Deleting a Message

The API should provide a way to delete SMS messages.

15
User Display Interface

The J2ME standard allows specific Canvas, Hardware Mapping and External Event Interaction functionality.

Canvas Functionality

The J2ME standard specifies that the Canvas class is available to the J2ME application.

The J2ME Canvas functionality controls the rendering of objects on the entire display. In addition, the J2ME Canvas requirements a minimum display size and functional soft key area for use by the application.

- There is a minimum of 96 x 54 pixels screen size to be available to the J2ME application.
- The lower 10 pixels are reserved for two soft keys and the menu icon. These 10 pixels are not counted in the 54 pixels reserved for the J2ME Canvas display.
- The soft keys and menu icon, located in the 10 pixel reserved area, are rendered in 11 high font.
- The soft keys and menu icon follow noted below:
    - MIDlet UI design can be designated in the J2ME Style Guide or Developer Guide but implementation is still decided by the developer.

The application supports the Portable Network Graphic (PNG) Image Format.

- Implementations are required to support images stored in the PNG format. All of the 'critical' chunks specified by PNG must be supported, with the following considerations:
    - The IHDR chunk. MIDP devices must handle the following values in the IHDR chunk:
        - All positive values of width and height are supported; however, a very large image may not be readable because of memory constraints.

- All color types are supported, although the appearance of the image will be dependent on the capabilities of the device's screen. Color types that include alpha channel data are supported; however, the implementation may ignore all alpha channel information and treat all pixels as opaque.
- All bit depth values for the given color type are supported.
- Compression method 0 (deflate) is the only supported compression method. This is the same compression method that is used for jar files, and so the decompression (inflate) code may be shared between the jar decoding and PNG decoding implementations.
- The filter method represents a series of encoding schemes that may be used to optimize compression. The PNG spec currently defines a single filter method (method 0) that is an adaptive filtering scheme with five basic filter types. Filtering is essential for optimal compression since it allows the deflate algorithm to exploit spatial similarities within the image. Therefore, MIDP devices must support all five filter types defined by filter method 0.
- MIDP devices are required to read PNG images that are encoded with either interlace method 0 (None) or interlace method 1 (Adam7). Image loading in MIDP is synchronous and cannot be overlapped with image rendering. There is no advantage for an application to use interlace method 1. Support for decoding interlaced images is required for compatibility with PNG and for the convenience of developers who may already have interlaced images available.

o The PLTE chunk. Palette-based images must be supported.

o The IDAT chunk. Image data may be encoded using any of the 5 filter types defined by filter method 0 (None, Sub, Up, Average, Path).

o The IEND chunk. This chunk must be found in order for the image to be considered valid.

o Ancillary chunk support. PNG defines several 'ancillary' chunks that may be present in a PNG image but are not critical for image decoding. A MIDP implementation may (but is not required to) support any of these chunks. The implementation should silently ignore any unsupported ancillary chunks that it encounters. The defined ancillary chunks are:

- bKGD cHRM gAMA hIST iCCP iTXt pHYs sBIT sPLT sRGB tEXt tIME tRNS zTXt

o All MIDlets will be displayed in the ZOOM OUT mode. Once the MIDlet is exited, the handset will return to the user’s pre-defined ZOOM IN/OUT state.

Hardware Mapping

The J2ME standard provides a minimum key set that is determined to be available on all devices. These key sets are to be mapped by the individual hardware manufacturers to a specified command set for application use. In addition, if other hardware keys are available, it is left to the manufacturer to determine the functionality of these keys. The following sections detail the standard J2ME required keys and the Motorola proprietary reserved key sets.

J2ME Standard Key Set

If present on the device specific hardware, the following keys are to be available to the J2ME application. The J2ME application can assign actions to these key inputs per the design of the application.

- Keys 0 through 9, the * key and the # key shall be available to the application for use. The J2ME application can either retain the standard functionality (i.e. a key press of 6 indicates the use of the number value six) or the application can map these keys to application specific use (i.e. the number 6 could map to a right action key).

Motorola Standard Key Set

The following keys are not available to the J2ME application and are retained by the handset for Motorola specific usage regardless of J2ME application processing. These keys and associated actions will always retain a higher priority than the J2ME application. This will ensure that some minimal level of control is always available to the handset user, regardless of the actions of a J2ME application.

End Key

The End key shall always terminate the current J2ME application and return the user back to the mobile idle state.

Send Key

The Send key shall always remain independent of any J2ME applications. When a telephone number from a J2ME application is displayed, the handset shall attempt to place the call if the Send key is pressed.

Left, Right and Menu Soft Keys

The J2ME standard allows for a global command set to be available to the application developers. This complete command list is: SCREEN, BACK, CANCEL, OK, HELP, STOP, EXIT, MENU AND ITEM.
For all other global commands, the handset shall use the default label (which will be the same name as the command) if a label is not supplied. If a command label supplied by the J2ME application is too long, it should be truncated.

The handset controls the rendering and positioning of these global commands, but the J2ME application determines the actual availability and functionality of these commands. For example, J2ME application1 may indicate to the KVM that only four of the global commands are to be used/displayed, where as J2ME application2 may indicate that all nine global commands are to be used/displayed.

Each of global commands in Table 17 has a priority ranking.

Global Command	Soft Key Position	Priority
SCREEN	RIGHT	1
BACK	LEFT	2
CANCEL	LEFT	3
OK	RIGHT	4
HELP	RIGHT	5
STOP	RIGHT	6
EXIT	LEFT	7
OPTIONS (menu)	RIGHT	8
ITEM	RIGHT	9
FIRE	RIGHT	10

Table 17 Key Ranking Priority

Rules:

- If no global command is specified from the J2ME application, the handset shall always display the BACK command on the left soft key position. The MENU icon and Right soft key shall remain blank. If the BACK key is pressed, the J2ME application is destroyed and the handset shall return back to the J2ME Application Manager screen.
- If the global command list is populated, but does not contain one of the following commands, BACK / CANCEL / EXIT, then the handset shall always display the BACK command on the left soft key position. If the global command list is populated with commands that occupy both left and right side soft keys, then the highest priority command shall occupy the appropriate softkey. The remaining global commands will be displayed if the menu soft key is pressed. The remaining global commands shall be displayed in ascending order based on the priority value of the above table.

Example 1:

Upon start-up, the J2ME application indicates that the following global commands are active: BACK, CANCEL, OK, HELP and STOP. The handset display would look like this:

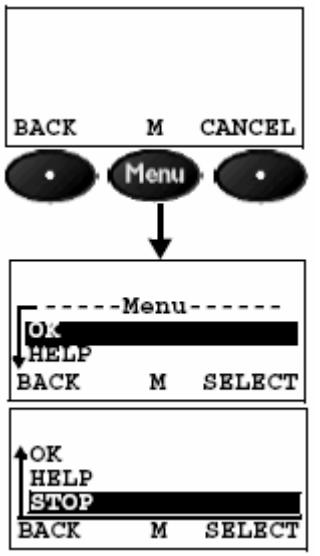


Figure 2 Active Global Commands - Back, Cancel, OK, Help & Stop

Example 2:

Upon start-up, the J2ME application indicates that the following global commands are active: SCREEN, CANCEL, HELP, OK, and STOP. The handset display would look like this:

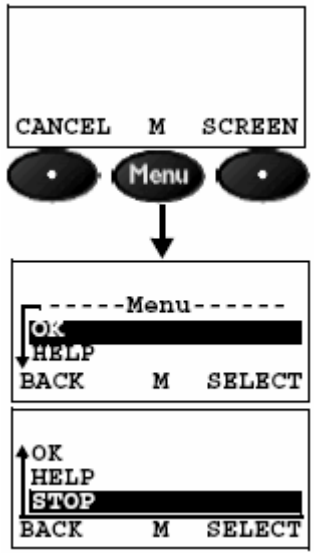


Figure 3 Active Global Commands - Cancel, Screen, OK, Help & Stop

Power Key

When the Power key is pressed and held down, it should always override all applications and power down the handset.

Volume Key

Volume keys will not be supported in the first release of J2ME, Java 1.0. In following releases, when the Volume key (s) is pressed and held down, it should always control the speaker/ accessory volume regardless of application.

Game Key Mapping

Games require the use of UP/DOWN/LEFT/RIGHT and FIRE as a basic set of keys in order to play. For handsets that have a four direction navigational device (joystick), UP/DOWN/LEFT/ RIGHT should be mapped to that. Otherwise, the keypad should return game action by mapping the 2 key to UP command, 8 key to DOWN command, 4 key to LEFT command, and 6 key to RIGHT command. In addition, the FIRE key should be mapped to the SELECT key if present otherwise to the right soft key and the 5 key. Additional game action keys exist, these are the GAME A, GAME B, GAME C, and GAME D keys. They perform different game actions depending on the game being played. GAME A should be mapped to the 1 key. GAME B should be mapped to the 3 key. GAME C should be mapped to the 7 key. GAME D should be mapped to the 9 key. For hardware implementations that support multiple key presses, such as Talon, the following keys must be supported when pressed simultaneously: Four direction navigational device (i.e. joystick), 1, 3, 5, 7, & 9 keys.

16
One-Click Application Access

The one-click access to J2ME applications Implementation allows the user to more readily access frequently used Java applications via the soft keys, navigation keys or smart keys. The key requirement is to allow MIDlets to behave as if they were Main Menu applications. Allowing J2ME application to be moved from the My JavaApps menu to the Main Menu, although this would be ideal, it will not be implemented until native applications are capable of being moved up and down the menu structure in different 'folders' and 'sub-menus'.

Application Resources

The DRM application resources table must be updated to provide for the display of the following.

- Small icon (no deletion)
- Large icon (no deletion)
- Softkey name
- Common name

Application Keys

J2ME applications must be able to be assigned to the application keys of the device. The options for application keys will be dependent on the device being used. This includes devices that have Smartkeys or four-way navigation keys that can be associated to applications. The applications executed from the application keys must also be accessible from the My JavaApps menu.

Application Icons

J2ME applications must be representable via icons of multiple sizes. This is especially required for devices that have multiple methods for displaying applications.

- Large Icons - Until a better solution can be implemented, a generic large animated icon must be created for display on the idle screen.
- Small Icons - A 15 x 15 pixels (Must be used in the Personalize list views).
    - The MIDlet-icon JAD attribute specifies the directory name for the icon(PNG image)
    - MIDlets with the MIDlet-icon attribute have the capability to display a PNG image
    - Next to the MIDlet name in the personalize list view.
    - MIDlets that do not have this attribute will have a standard PNG image displayed
    - Next to the MIDlet name.

Softkey Labels

The softkey name of the app must be truncated (if the name is too long) after all available space is used. In the rare occasion that there are two applications with the same name (ex. two calculator apps: basic and advanced), the second application name must be truncated followed by the number ‘2’).

Effect of Master Clear or Master Reset

The one-click keys that are associated to a J2ME application must be replaced by default settings if a Master Clear or Master Reset is performed.

Deleting the MIDlet/Application

Deleting the J2ME MIDlet/application from the My JavaApps list must reset the application keys to their defaults.

17 Download 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 user may 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.

Star Active Browser Session from Main Menu

The Figure 4 describes Staring Active Browser Session from Main Menu:

Figure 4 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 J2ME 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 J2ME Application

Once connected to the WAP browser, different locations may be visited where J2ME 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 will be Mot-Data-Space-Requirements and Mot-Program-Space-Requirements. These attribute will 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 will be displayed before the download begins.

Downloading MIDlets

The Figure 5 represents J2ME Application (MIDlets) Download and Installation.

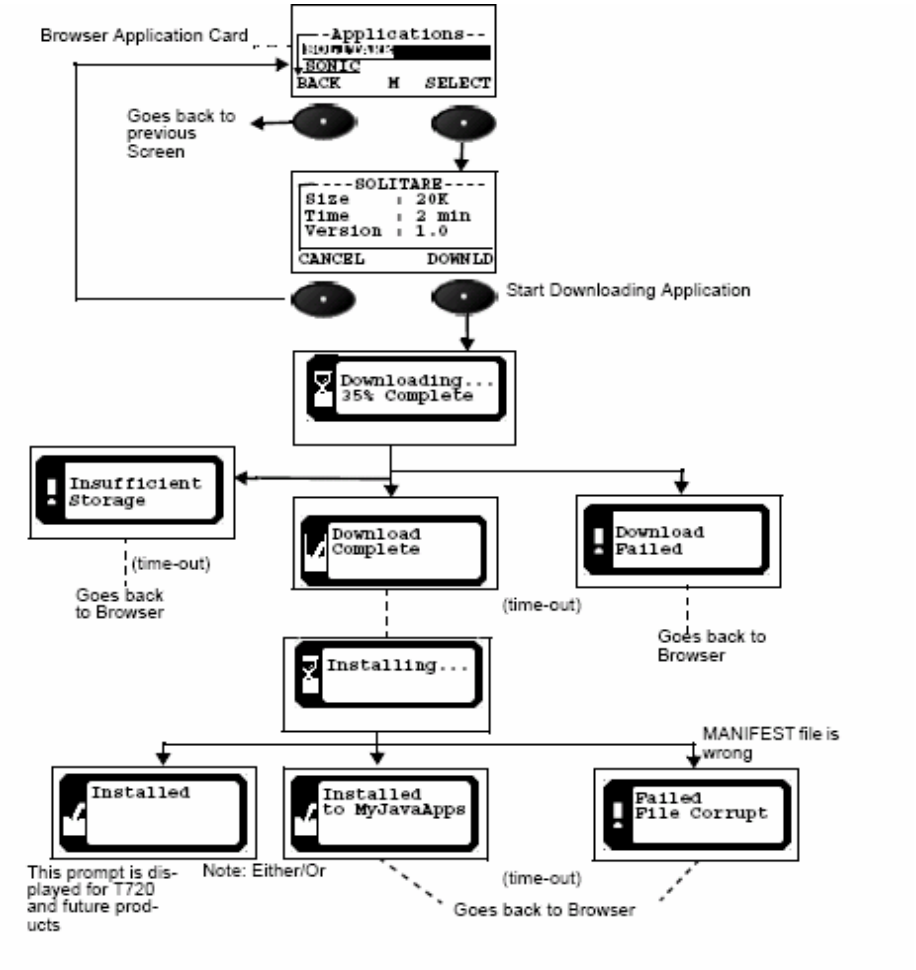


Figure 5 Downloading and Installing J2ME Application (MIDlets)

Steps to Download and Install J2ME Application:

- BACK shows previous screen to the user.
- If the SELECT softkey is selected, the handset shows display 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 will be 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 will be unable to send the HTTP POST with the MIDlet-Install Notify attribute. In this case, the MIDlet will be deleted to insure 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 will not be in the needed state in order to return the MIDlet Install Notify attribute.
- Upon completing Installation, the handset displays a transient notice “Installed to Games & 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 will continue and the handset will not return to the idle display. When the flip is opened, the “Installing...” dialogue should appear on the screen and should be dynamic.
- During download or install of application, voice record, voice commands, voice shortcuts, and volume control will not be 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.

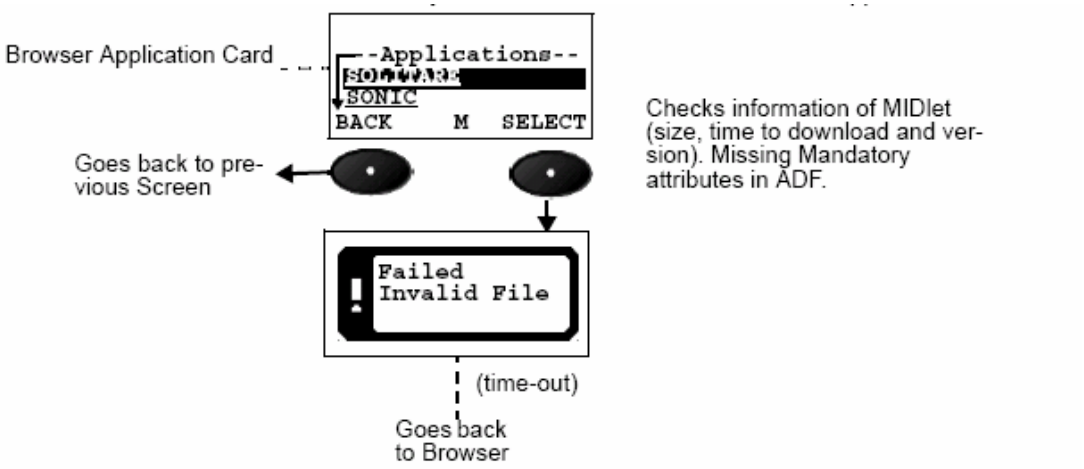


Figure 6 Application does not have Mandatory Attributes in ADF

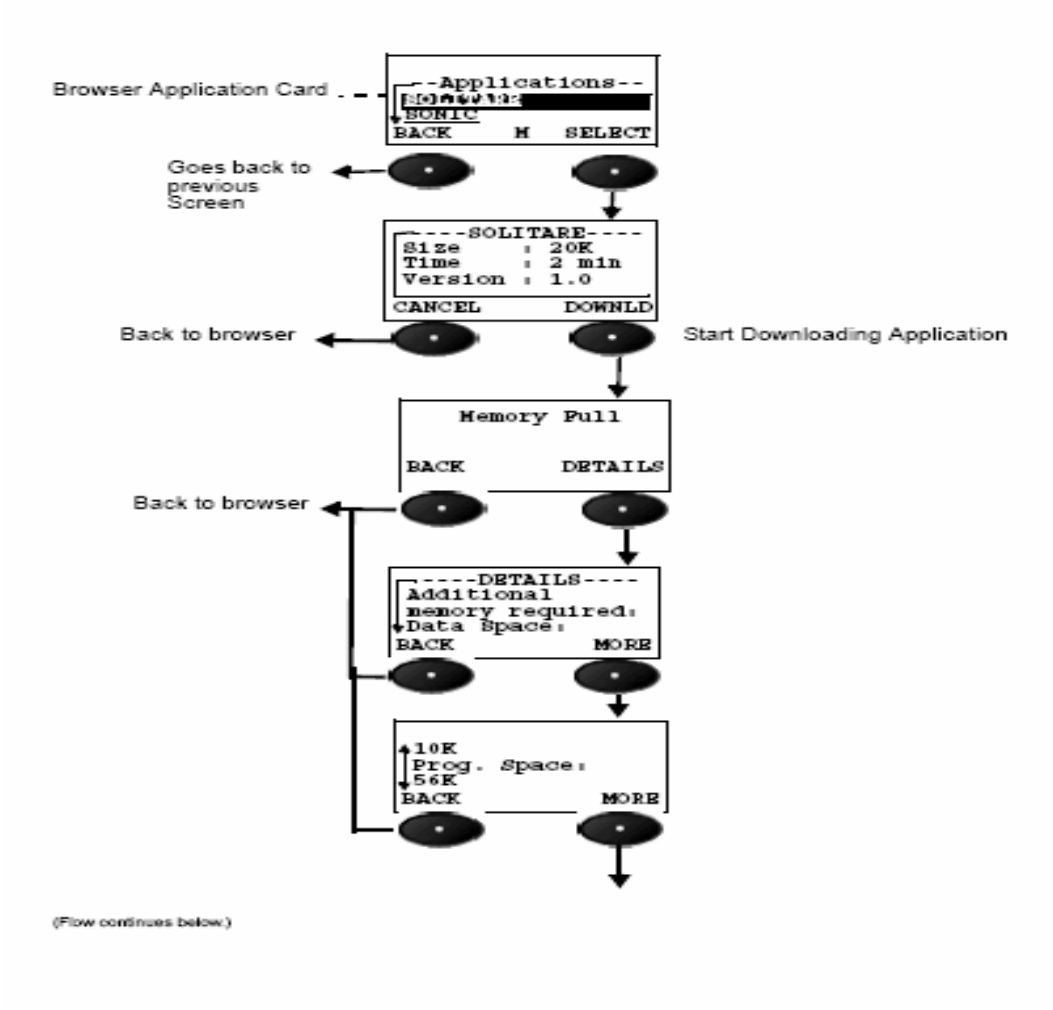
Different Error Checks

Memory Full

There are two distinct cases when a Memory Full error can occur during the download process. Memory Full will be 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 will be 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 in order to free additional memory. The following messages and screen flows will be displayed depending on whether specific JAD attributes are present or not:



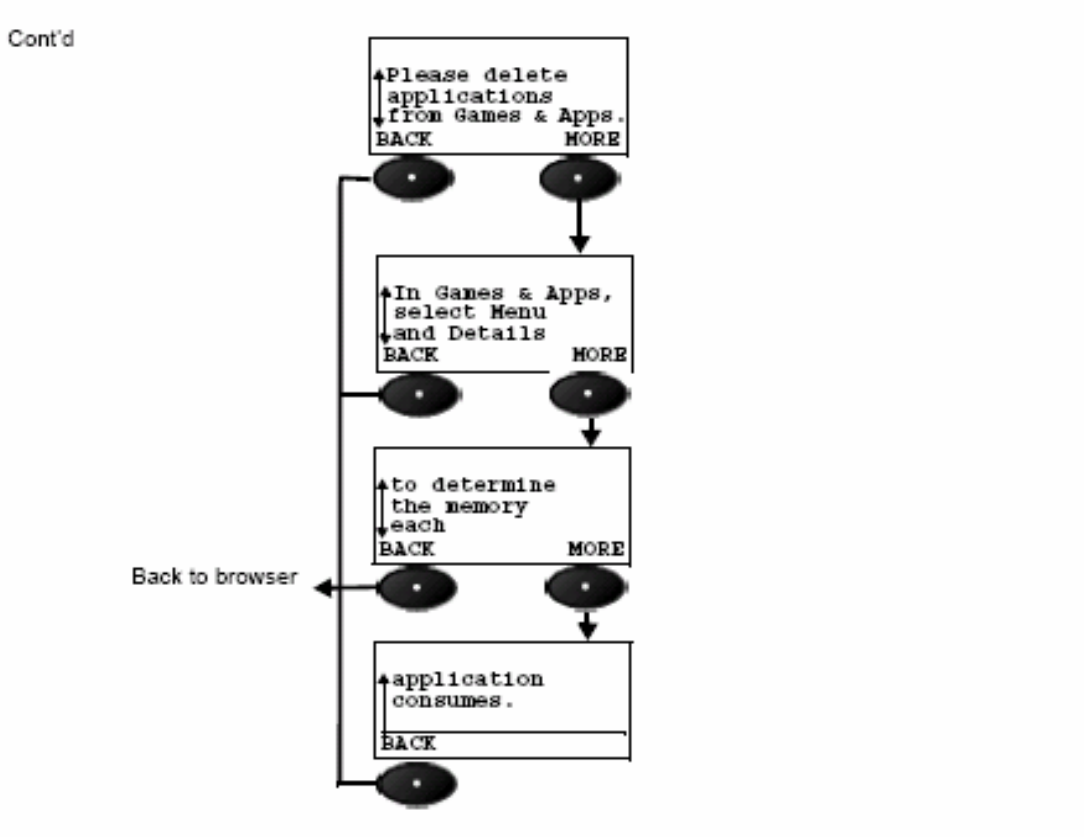


Figure 7 Memory full error

Rules:

- If Mot-Data-Space-Requirements and Mot-Program-Space-Requirements attributes are present in the JAD, the above noted prompt should be displayed. This value takes into account the memory requirements of the MIDlet and the current memory usage on the phone, in order 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 will no longer be a transient prompt but a dialog screen with a Help softkey and a Back softkey will be displayed.
- DETAILS will give the user the above detailed Help screen describing the memory required to be able to download the MIDlet.
- The Help dialog will include 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 will take the user back to the browser page from which the user selected the MIDlet to download.

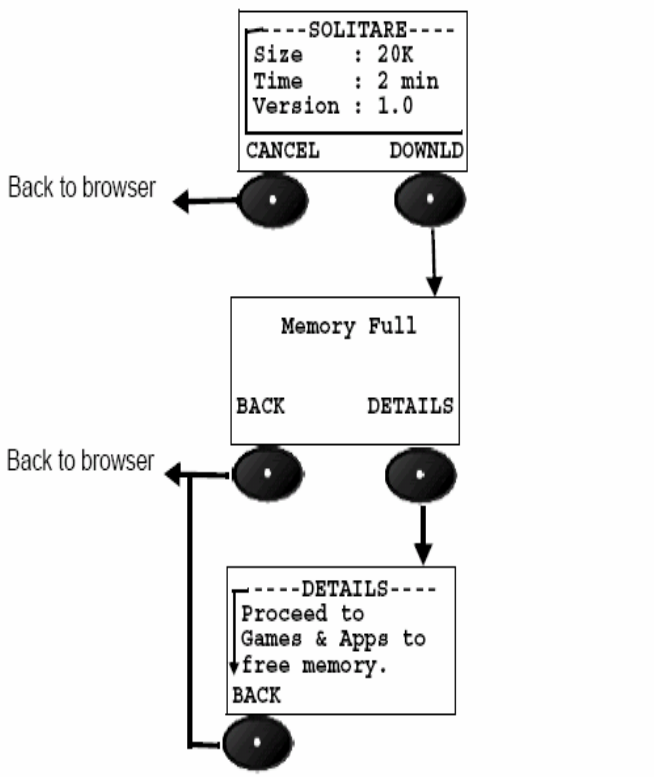


Figure 8 Mot-Data-Space & Mot-Program-Space attributes are not present or are incorretct

Rules:

- 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 will display the above help dialog.
- The Help dialog will include 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 9 must be displayed:

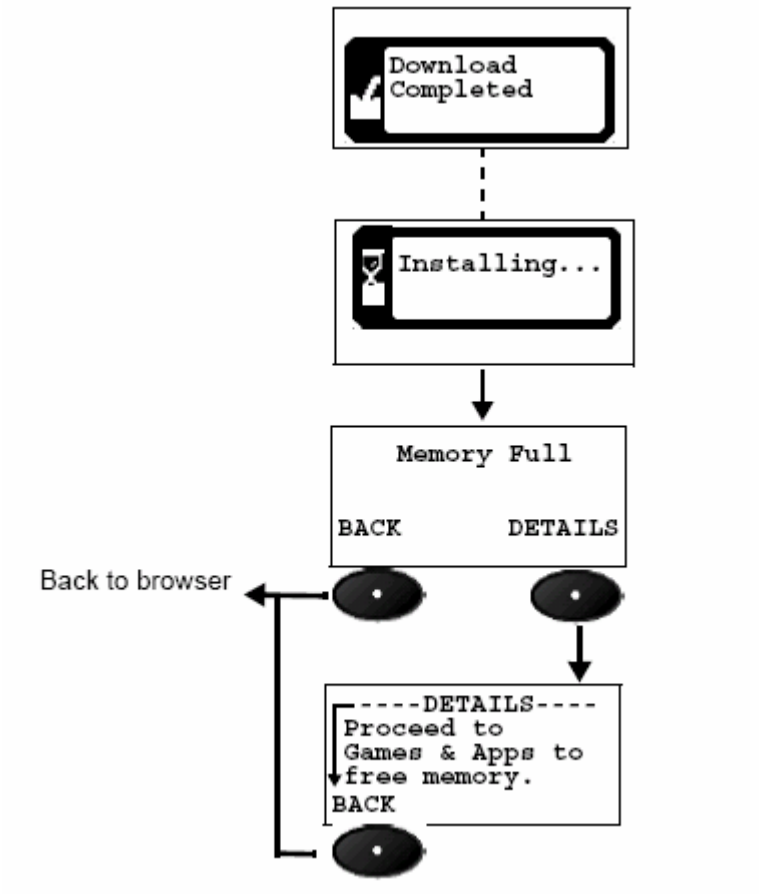


Figure 9 Memory Full help message during installation process

Rules:

- The installation process is canceled when this error condition occurs.
- The Memory Full error will no longer be a transient prompt but a dialog screen with a Help softkey and a Back softkey will be displayed.
- DETAILS will give the user the above Help screen explaining that additional memory is required to be able to install the MIDlet.
- The Help dialog will include 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 will take 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.

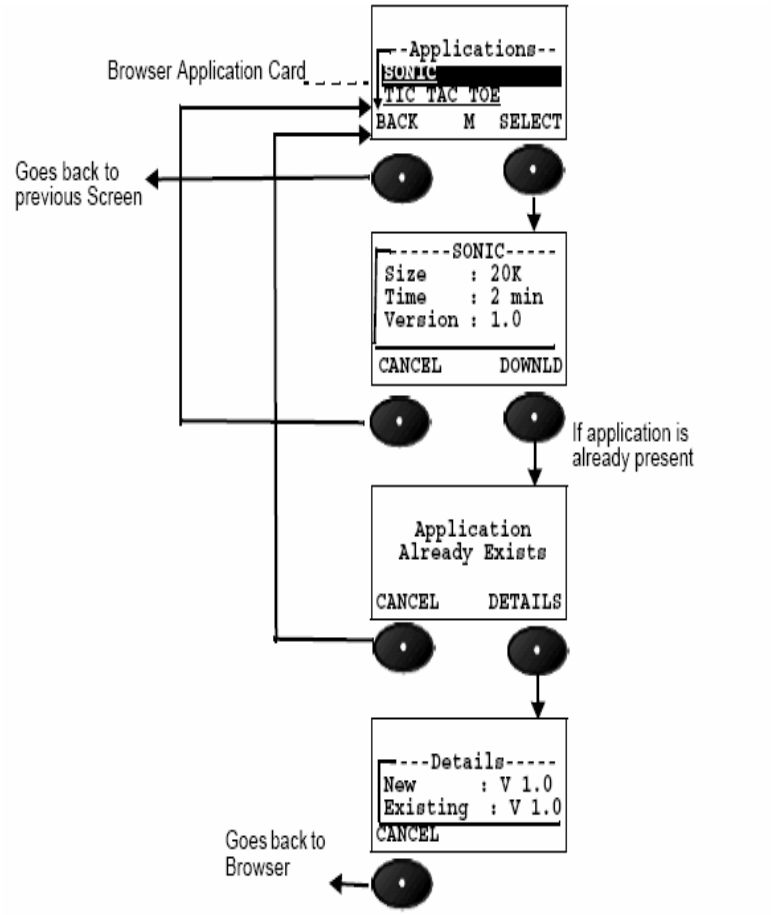


Figure 10 Same Version of Application already exists on the handset

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.

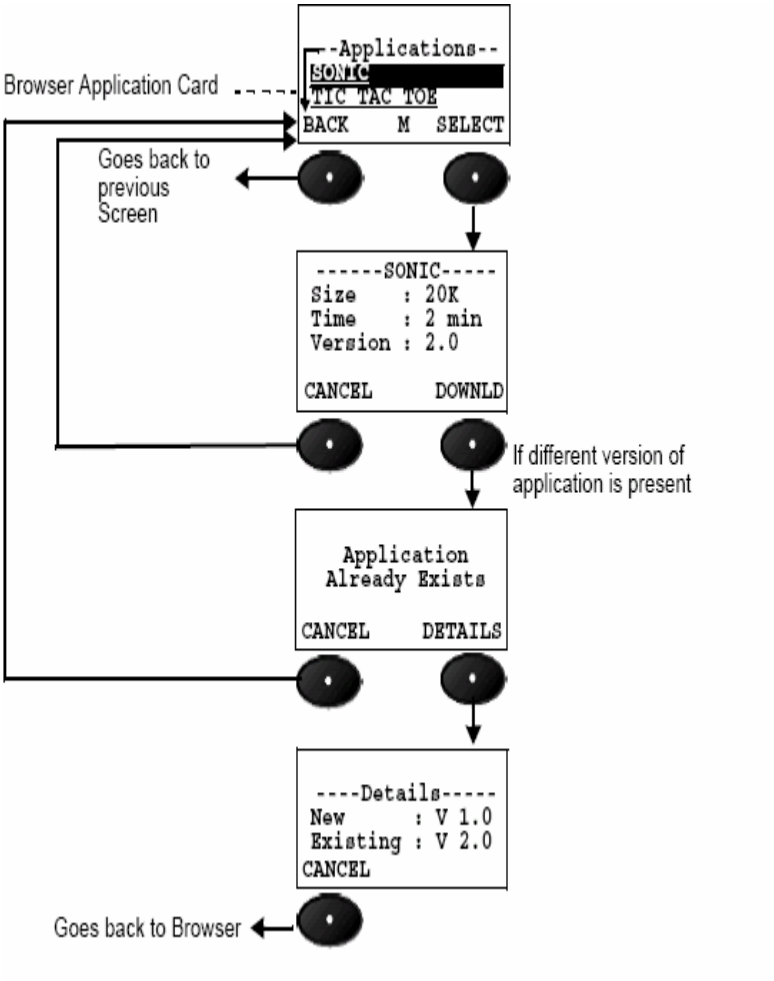


Figure 11 Latest (Newer) Version of Application exists

Rules:

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

18 Lightweight Windowing Toolkit

LWT integrate with the LCDUI API within the MIDP and enhance the capabilities to include a component-level API through which developers can control the contents and layout of their screens. These components are graphical user interface elements such as buttons, check boxes, text fields, images, etc. LWT also allow developers to create new imaginative components or change the look and feel of existing components.

Those MIDlets taking advantage of the added capabilities of LWT will only run on those handsets that incorporate the LWT LOC.

LWT extends the MIDP class hierarchy by extending the LCDUI Canvas class. The ComponentScreen is a subclass of Canvas, which means it can be easily added to a MIDP implementation and minimize dependencies and maintenance overhead. This also allows standard MIDlets to mix both MIDP screens and LWT screens in their MIDlets.

LWT is designed to use MIDP low-level graphics routines exclusively, which adds to the ease of implementation. Although no device-specific modifications are required, an LWT implementation may be tailored to match the rest of the device's user interface. The text components in LWT may be integrated with the device's data entry mechanisms, such as handwriting recognition or predictive keypad input.

LWT implementation requires the handset to be MIDP compliant and have approximately 30KB of flash memory.

LWT does not expose any additional native interfaces and only relies on mechanisms specified by MIDP 1.0. LWT can be safely exposed to untrusted MIDlets.

Once LWT is implemented, the LWT TCK must be completed and pass successfully.

19
UDP Support

This functionality is to enable J2ME applications access to Generic UDP Transport Service.

- This enhancement allows for J2ME applications to utilize the UDP header compression format for data applications over IP.
- The API should follow the guidelines of the User Datagram Protocol standard, IETF RFC 768, J. Postel, August 28, 1981.
- This functionality should be available for both CSD and GPRS connections.

20 Shared JAD URLs

Overview

Actually, users are able to download J2ME applications. The first step is to download the JAD file and, after a confirmation, the site is launched to download the application. If they want to forward the JAD link to someone else, it's impossible.

The Share JAD URLs is a feature that resolves the prior problem, it allows users to share their downloaded J2ME application URLs with others. When J2ME applications are downloaded, the browser shall provide the Java Application Manager (JAM) with the JAD URL address. When J2ME applications are downloaded via PC or MMS, a new JAD attribute shall specify the JAD URL address.

Tell-A-Friend Option

When entering the J2ME application context-sensitive menu, a Tell-A-Friend option will be provided. Upon selecting this option, the standard SMS messaging form will appear. The link to the URL where the application JAD file can be found and its name will be pre-populated into the message body. This allows the user to send messages to friends, telling them where to download the application.

Upon receipt of a Tell-A-Friend message, a Motorola handset user should be able to use the browser's GOTO functionality. Selecting GOTO will cause the download of JAD to occur. The remaining download steps will occur as normal.

Accessing Tell-A-Friend from SMM

The MIDlet Manager menu lets the user perform certain actions on the selected MIDlet suites. The Table 18 describes the various actions that can be performed on a suite.

Action	Description
Tell-A-Friend	Populates a message with the link to the application's JAD URL inserted into the message body, following messaging standard behavior for pre-populated messages.
Details	Displays the information about the suite. This includes MIDlet suite name, vendor, version, number of apps in suite, flash usage, both data and program space of the application.
Delete	Lets the user delete a suite. A confirmation is requested before the application is deleted.

Table 18 Performed on a suite

Rules:

- If the application does not have an associated JAD URL, the Tell-A-Friend option will not appear in the context-sensitive menu.
- If the URL plus the application name size exceed the maximum size allowed for an SMS message, the following rules shall be applied, in this order, to truncate the link:
  - Remove application name from link, (i.e.: "GP http://www.mot.com/games/gp.jad" shall be truncated to "http://www.mot.com/games/gp.jad").
  - Remove path to the JAD file, keeping only server's URL and application name. (i.e.: "GP http://www.mot.com/games/gp.jad" shall be truncated to "GP http:// www.mot.com").
- If server's URL cannot be kept, Tell-A-Friend option shall be disabled.
- J2ME applications downloaded through a PC can specify JAD URL using new JAD attribute Mot-Midlet-URL. If this JAD attribute is present, JAM will use the JAD URL specified by this attribute to enable the Tell-A-Friend option.
- J2ME applications downloaded through MMS (when this functionality is available on the phone) can specify JAD URL using new JAD attribute Mot-Midlet-URL. If this JAD attribute is present, JAM will use the JAD URL specified by this attribute to enable Tell-AFriend option.
-

The Figure 12 illustrates the MIDlet Manager and the context-sensitive menus:

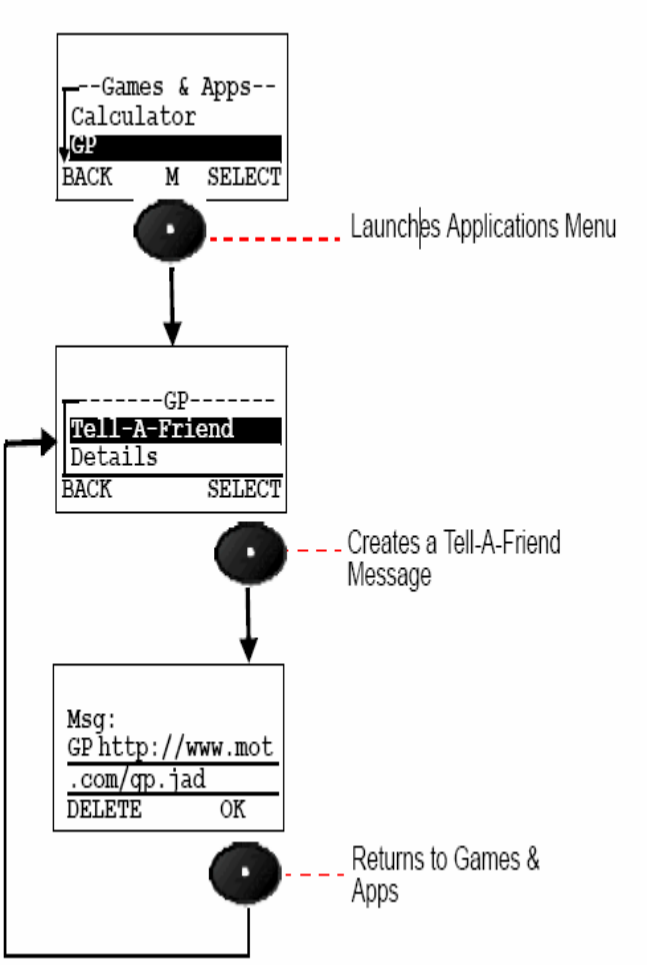


Figure 12 the MIDlet Manager and the context-sensitive menus

Downloading through Browser

Rules:

- When downloading a J2ME application, the browser shall provide JAM with the URL of the JAD file. This URL shall override the URL specified by Mot-Midlet-URL attribute in case of conflict.

Downloading from PC (Via serial/USB)

Rules:

- When downloading an application, JAM shall use the JAD attribute Mot-Midlet-URL if present to get the JAD URL.

Downloading through MMS

Rules:

- When downloading an application via MMS, JAM shall use the JAD attribute Mot-Midlet- URL if present to get the JAD URL.

21 Get URL from Flex API

Overview

This feature allows accessing URL 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. So, this feature would allow Java applications to read and display the URL stored in flex for users that would like to download new levels of Game.
The existing functionality allows current Java Applications use a dedicated URL to inform users about the location which a new level of game can be downloaded. This new functionality allows carriers to specify the URL for content download.

Flexible URL for downloading functionality

The URL is flexed using RadioComm or using OTA provisioning. The URL will follow the rules mentioned below:

- All URLs used shall follow the guidelines outlined in RFC1738: 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 flex table. The default URL may be “http://www.hellomoto.com”.

The Java Application will be able to access the flexed URL by System.getProperty method. The key for accessing the URL is “com.mot.carrier.URL”. The method System.getProperty will return NULL if no URL is flexed.

Security Policy

Only trusted applications will be granted permission to access this property.

22
Multiple Key Press

Multi-button press support enhances the gaming experience for the user. Multi-button press support gives the user the ability to press two (2) keys simultaneously and the corresponding actions of both keys will occur simultaneously. An example of this action would be the following:

- If Left + Fire were pressed at the same time, the Java object (e.g Canvas) will receive Left Pressed + Fire Pressed. In the same way, when the 2 keys are released, Java object (e.g. Canvas) will receive Left Released + Fire Released.

The following sets of keys will support multi-button press support on the Motorola C381p handset. Multi-button press within each set will be supported, while multi-button press across these sets or with other keys will not be supported.

Set 1 – Nav (Up), Nav (Down), Nav (Right, Nav (Left)

Refer to the Table 19 for gaming and keypad feature/class support for MIDP 2.0:

Feature/Class	Implementation
lcdui.game package	Supported
setBacklight as defined in javax.microedition.lcdui.Display	Supported
setVibrator as defined in javax.microedition.lcdui.Display	Supported
All constructors and inherited classes for the IllegalStateException in java.lang	Supported
All constructors, methods, and inherited classes for the Timer class in java.util	Supported
All the constructors, methods, and inherited classes for the TimerTask class in java.util	Supported
All fields, constructors, methods, inherited fields and inherited methods for the GameCanvas class in javax.microedition.lcdui.game	Supported
Map the UP_PRESSED field in javax.microedition.lcdui.game.GameCanvas to the top position of the	Supported

key.	
Map the DOWN_PRESSED field in javax.microedition.lcdui.GameCanvas to the bottom position of the key	Supported
Map the LEFT_PRESSED field in javax.microedition.lcdui.GameCanvas to the left position of the key	Supported
Map the RIGHT_PRESSED field in javax.microedition.lcdui.GameCanvas to the right position of the key	Supported
All methods and inherited methods for the Layer class in javax.microedition.lcdui.game	Supported
All constructors, methods, and inherited methods for the LayerManager class in javax.microedition.lcdui.game.Layer	Supported
All fields, constructors, methods, and inherited methods for the Sprite Class in javax.microedition.lcdui.game Sprite Frame height will not be allowed to exceed the height of the view window in javax.microedition.lcdui.Layer	Supported Any, limited by heap size only
Sprite frame width will not be allowed to exceed the width view of the view window in javax.microedition.lcdui.Layer Sprite recommended size All constructors, methods, and inherited methods for the TiledLayer class in javax.microedition.lcdui.game	Any, limited by heap size only 16*16 or 32*32 Supported
MIDlet Queries to keypad hardware	Supported
Alpha Blending	Transparency only

Table 19 Gaming and keypad feature/class

23 iTAP

Intelligent Keypad Text Entry API

When users are using features such as SMS (short message service), or "Text Messaging", they can opt for a predictive text entry method from the handset. The J2ME environment has the ability to use SMS in its API listing. The use of a predictive entry method is a compelling feature to the MIDlet.

This API will enable a developer to access iTAP, Numeric, Symbol and Browse text entry methods. With previous J2ME products, the only method available was the standard use of TAP.

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

The following J2ME text input components will support iTAP.

- `javax.microedition.lcdui.TextBox`

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

- `javax.microedition.lcdui.TextField`

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

Refer to the Table 20 for iTAP feature/class support for MIDP 2.0:

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

Table 20 ITAP feature/class

24 LCDUI

LCDUI API

The Table 21 lists the specific interfaces supported by Motorola implementation:

Interface	Description
Choice	Choice defines an API for user interface components implementing selection from a predefined number of choices.
CommandListener	This interface is used by applications which need to receive high-level events from implementation
ItemCommandListener	A listener type for receiving notification of commands that have been invoked on Item[286] objects
ItemStateListener	This interface is used by applications which need to receive events that indicate changes in the internal state of the interactive items within a Form[231] screen.

Table 21 Interfaces supported by Motorola implementation

The Table 22 lists the specific classes supported by Motorola implementation:

Classes	Description
Alert	An alert is a screen that shows data to the user and waits for a certain period of time before proceeding to the next `Displayable`.
AlertType	The `AlertType` provides an indication of the nature of alerts.
Canvas	The `Canvas` class is a base class for writing applications that need to handle low-level events and to issue graphics calls for drawing to the display.
ChoiceGroup	A `ChoiceGroup` is a group of selectable elements intended to be placed within a Form.
Command	The `Command` class is a construct that encapsulates the semantic information of an action.
CustomItem	A `CustemItem` is customizable by sub classing to introduce new

	visual and interactive elements into `Forms`.
DateField	A `DateField` is an editable component for presenting date and time (calendar) information that will be placed into a `Form`.
Display	`Display` represents the manager of the display and input devices of the system.
Displayable	An object that has the capability of being placed on the display.
Font	The `Font` class represents fonts and font metrics.
Form	A `Form` is a `Screen` that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, choice groups, and custom items.
Gauge	Implements a graphical display, such as a bar graph of an integer value.
Graphics	Provides simple 2D geometric rendering capability.
Image	The `Image` class is used to hold graphical image data.
ImageItem	An item that can contain an image.
Item	A superclass for components that car be added to a Form[231].
List	A `Screen` containing a list of choices.
Screen	The common superclass of all high-level user interface classes.
Spacer	A blank, non-interactive item that has a settable minimum size.
StringItem	An item that can contain a string.
TextBox	The `TextBox` class is a `Screen` that allows the user to enter and edit data.
TextField	A `TextField` is an editable text component that will be placed into a Form.
Ticker	Implements a “ticker-tape”, a piece of text that runs continuously across the display.

Table 22 Specific classes supported by Motorola implementation

Refer to Table 23 for LCDUI feature/class support for MIDP 2.0:

Feature/Class	Implementation
All fields, constructors, methods, and inherited methods for the Alert class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, and inherited methods for the AlertType class in the javax.microedition.lcdui package	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of ALARM	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of ERROR	Supported

Will provide and play an audible sound when the play Sound() method is called with an AlertType of WARNING	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of CONFIRMATION	Supported
Will provide and play an audible sound when the play Sound() method is called with an AlertType of INFO	Supported
All fields, constructors, methods, and inherited methods for the Canvas Class in the javax.microedition.lcdui. package	Supported
Status indicators out of full-screen mode will consume a portion of the display	Supported
UP field in javax.microedition.lcdui.Canvas to the top position of the key	Supported
DOWN field in javax.microedition.lcdui.Canvas to the bottom position of the key	Supported
LEFT field in javax.microedition.lcdui.Canvas to the left position of the key	Supported
RIGHT field in javax.microedition.lcdui.Canvas to the right position of the key	Supported
All fields and methods for the Choice interface in the javax.microedition.lcdui package	Supported
Truncate an image in a Choice object if it exceeds the capacity of the device display	Supported
Truncation of very long elements will not occur in a Choice object	Text in forms is wrapped and scrolled
Will display a portion of long elements to display and provide a means for the user to view all of the parts of the element	Supported
Truncation in elements w/line breaks will not occur in a Choice object	Supported
Portion of line break elements to display and provide a means for the user to view all parts of the element	Supported
All constructors, methods, inherited fields, and inherited methods for the ChoiceGroup class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the Command class in the javax.microedition.lcdui package	Supported
All methods for the CommandListener interface in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the CustomItem abstract class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the DateField class in the javax.microedition.lcdui	Supported

package	
All fields, methods, and inherited methods for the Display class in the javax.microedition.lcdui package	Supported
Maximum colors for the numColors() method in javax.microedition.lcdui.Display	64K colors supported
All methods and inherited methods for the Displayable class in the javax.microedition.lcdui package	Supported
Adding commands to soft buttons before placing it in a menu for the addCommand() method in javax.microedition.lcdui.Displayable	Supported
All fields, methods, and inherited methods for the Font class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the FORM class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the Gauge class in the javax.microedition.lcdui package	Supported
All fields, methods, and inherited methods for the Graphics class in the javax.microedition.lcdui package	Supported
DOTTED stroke style	Supported
SOLID stroke style	Supported
All methods and inherited methods for the Image class in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the ImageItem class in the javax.microedition.lcdui package	Supported
All fields, methods, and inherited methods for the Item class in the javax.microedition.lcdui package	Supported
Label field	Supported
All methods for the ItemCommandListener interface in the javax.microedition.lcdui package	Supported
All methods ItemStateListener interface in the javax.microedition.lcdui package	Supported
All fields, constructors, methods, inherited fields, and inherited methods for the List class in the javax.microedition.lcdui package	Supported
All constructors, methods, inherited fields, and inherited methods for the Spacer class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the StringItem class in the javax.microedition.lcdui package	Supported
All constructors, methods, and inherited methods for the TextBox class in the javax.microedition.lcdui package	Supported

All fields, constructors, methods, inherited fields, and inherited methods for the TextField class in the javax.microedition.lcdui package	Supported
Visual indication with UNEDITABLE field set will be provided	Supported
All constructors, methods, and inherited methods for the Ticker class in the javax.microedition.lcdui package	Supported
OEM Lights API providing control to the lights present on the handset	Supported, Fun Lights API
All fields, constructors, methods, inherited fields, and inherited methods for the TextField class in the havax.microedition.lcdui package	Supported

Table 23 LCDUI feature/class

25
Auto Launch of Midlets

The Java framework on the device must provide a mechanism to automatically launch midlets based on specific message. The message may be generated by several clients including a push message coming from the network. Once the KVM received the message, it takes the appropriate action of launching the midlet the message is intended for.

Scenarios involved in launching midlet

There are three possible scenarios when KVM has to pass the message to a midlet.

- The midlet is currently actively running: In this case, the KVM simply passes the message to the midlet. The client originating the request is notified that the midlet is already running.
- No midlets are active: In this case, the KVM must launch the midlet the message is intended for. Upon successfully starting, KVM should inform the client that the launch has been successful.
- The KVM is currently busy either running another midlet or installing another midlet. Since the KVM will support only one midlet at a time, the client originating the message should be notified that KVM is currently running application. The client then has the option of having KVM terminate the current midlet and launching another one.

NOTE: If the installation is terminated, appropriate cleanup actions should be performed. The midlets that were being installed should be left at an uninstalled state.

26 Background Applications

Background Attribute

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

Background Java Application Lifecycle

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

For example:

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

Background MIDlet

When a MIDlet with background attributes is running, the user can press the END (red) key to initiate the following options shown in Figure 13:

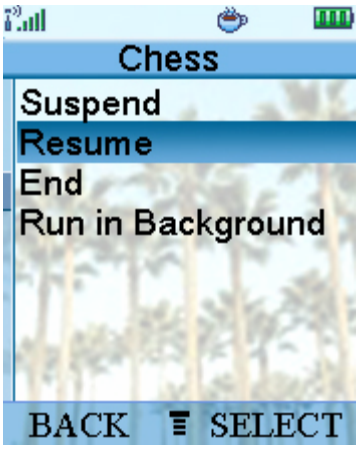


Figure 13 Java service menu for a MIDlet with background attributes

Pressing the END key will force the handset to display a Java service menu with the options listed in Figure 13.

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

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

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

Flip Behaviors

A Motorola specific JAD attribute called FlipInsensitive exists. When a MIDlet is running and the flip is closed by the user, the MIDlet will follow one of the following behaviors:

- Suspend – if the FlipInsensitive attribute is = false.
- Continue running – if the FlipInsensitive attribute is = true. In this case, audio resources will be available for the MIDlet.

27
Java System Menu

The Java System menu allows the user to see what version of MIDP and CLDC is being used in the phone. It also shows the user the free data space available, program space available, and the heap size being used. The Table 24 describes each function in detail.

Action	Description
CLDC Version	This displays the CLDC version that is being used in the MS.
MIDP Version	This displays the MIDP version that is being used in the MS.
Data Space	This displays the amount of free memory available for data used by the applications, i.e. phone book entries, game high scores.
Program Space	This displays the amount of free memory available for applications.
Heap Size	This is the amount of runtime memory available in the phone.

Table 24 Function Describes

MIDlet Manager Menu

The MIDlet Manager menu lets the user perform certain actions on the selected MIDlet suites. The table 25 describes the various actions that can be performed on a suite.

Action	Description
Details	Displays the information about the suite. This includes MIDlet suite name, vendor, version, number of apps in suite, flash usage, both data and program space of the application.
Details	Lets the user delete a suite. A confirmation is requested before the application is deleted.

Table 25 Midlet Manager Menu Description

View MIDlet Suite Information

To view information on any MIDlet suite, the user brings up the MIDlet Manager menu. The user highlights the Details option and SELECTs it to view the information. The information presented includes MIDlet suite name, vendor, version, number of apps in suite, and flash usage, both data and program space. The figure 14 illustrates this:

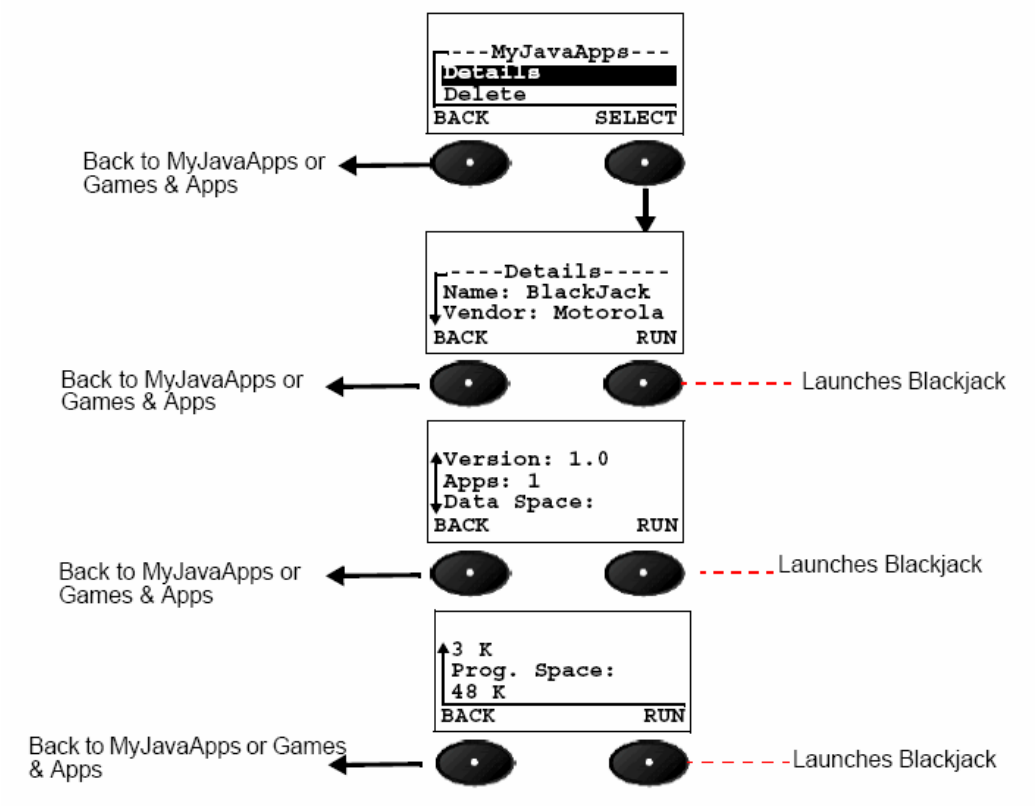


Figure 14 Viewing MIDlet Suite Information

Rules:

- The size information is the size taken up on the device. This size may be different from the JAR file size due to ROMizing.
- The details of the applications are shown in a Synergy Text field. For future releases, the title of the text field is the application name, however, for the current release it will show in the Figure 14. The information shown in the text fields are MIDlet suite name, vendor, version, number of apps in suite, data space, and program space. The name, vendor, version, number of apps in the suite will be shown on a separate line in the text field. The data space and program space will each use two lines to show the title on one line and the value on the next line.

Deleting MIDlet Suites

To delete any suite, the user brings up the MIDlet manager menu for the suite.The user

highlights the Delete option and SELECTs it to delete the suite. This brings up a confirmation
dialog. Upon confirmation by the user, the suite and all persistent data is deleted from the device. The figure 15 illustrates this.

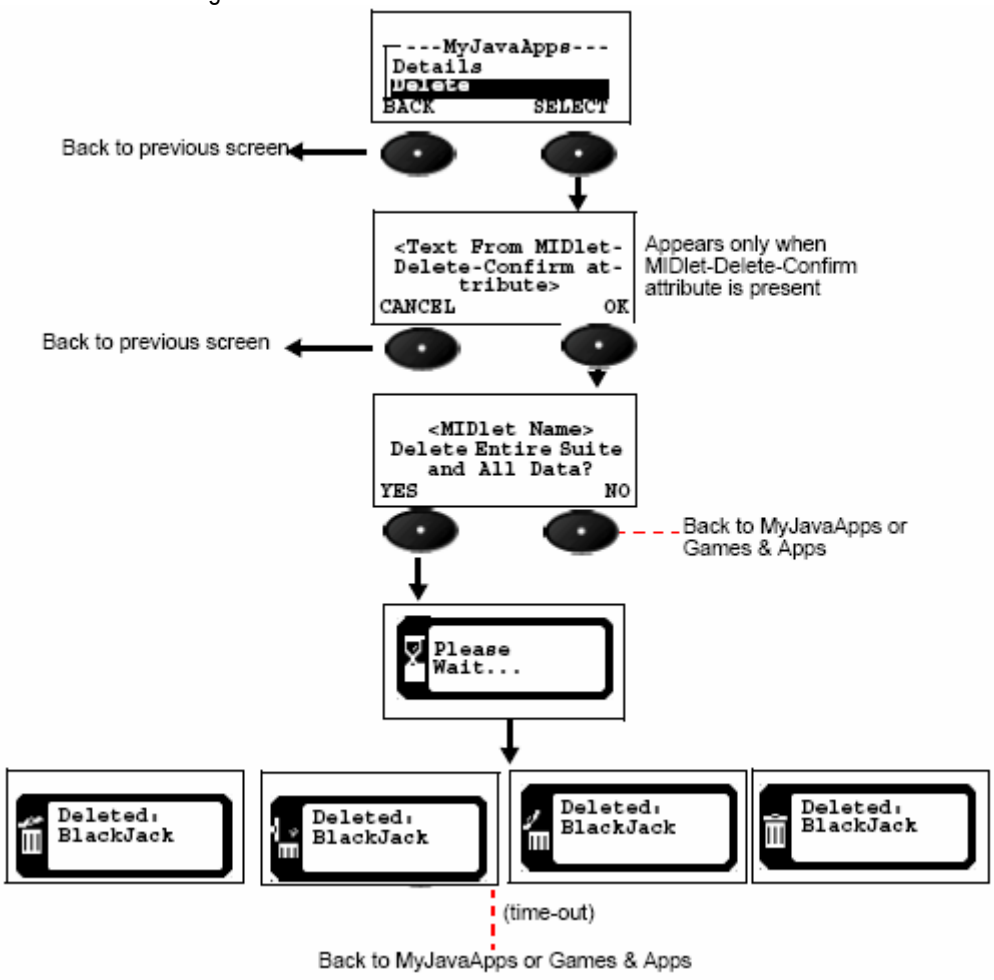


Figure 15 Deleting MIDlet Suites

Rules:

- If user selects YES, the suite, including all applications contained in the suite, and all persistent data must be deleted from the device and the menu item must be removed. The user is returned back to Main Menu.
- The text from the MIDlet's JAD Attribute 'MIDlet-Delete-Confirm' MUST be displayed. The text may be truncated to 70 characters. Also note that this text cannot be translated. If the 'MIDlet-Delete-Confirm' attribute is not present, this information will not be displayed.
- IF the user selects Cancel, the delete operation is not performed and the user is returned back to the previous screen.
- If the user selects OK, the confirmation dialog is displayed.

- If the application name is longer than 12 characters, it must be truncated. The “Delete Entire Suite and All data?” text should be displayed after the above information.
- While the MIDlet suit is being deleted, a dialog box will display, “Please Wait...” The dialog box will remain on the display until the application has been completely delete
- After deleting an application, a transient dialog will be displayed indicating to the user that the file and all its contents have been deleted. The transient delete animation dialog is displayed with the text, “Deleted:”, Newline, and the application name.
- The pre-installed applications will be dependent on handset type, carrier and region requirements.
- Pre-Installed MIDlet Suites will contain a JAD attribute called MIDlet-PreInstalled.
- If MIDlet-PreInstalled: TRUE and the Flex bit is set, then the MIDlet can not be deleted by the user

28
Invisible Net for J2ME

Introduction

This chapter presents the Invisible Net for J2ME feature for multi line graphics displays. By making it quicker and easier for the user to launch the browser for the embedded applications, the carrier can experience ARPU bumps due to the additional Internet traffic. Other benefits to customers and carriers include the simplification of the UI, elimination of the "hunt and peck" scenario when trying to access URLs across applications, and additional value-add in content/service delivery.

J2ME Invisible NET Options

J2ME applications can embed "Invisible Net" URLs within the J2ME components or within J2ME context-sensitive menus, such as Games & Apps, and Java Tools, as listed in the following sections.

J2ME Component Options

A product utilizing the Motorola J2ME solution should be able to use the "Invisible Net" functionality to embed URLs within the following J2ME menus:

- Games and Apps menu
- Java Tools menu
- HTTP downloads operation for J2ME.

Rules:

- The associated right softkey when a URL will be launched shall be GO TO.

- URLs associated with J2ME menus or components will be launched using one of the following methods:
    - HTTP download with the following exceptions:
        - Launch shall be directly to the URL associated with the J2ME menu item.
        - If flexed on, retrieve the URL from the HTTP client.
        - If flexed off, retrieve the URL through the browser.
        - Title can be flexible, and URL can be flexible.
    - If existing Web Sessions is used, launch shall be using the information from the default Browser session:
        - Instead, launch shall be directly to the URL associated with the J2ME menu item.
    - For future implementation, if Web Sessions Redesign has been implemented, launch of the URL will use the following:
        - Default Java Session parameters (or in the absence of a Java Session, the default Browser application settings) and the launch shall be directly to the URL associated with the Java menu item.
        - Default data connection parameters
        - Refer to MRS 7535 - Internet Setup (Web Sessions Redesign) for more information on application and data connection session parameters.
    - Upon exiting the active browser or HTTP session, the user shall be returned to the Java menu from which the browser was launched from. (Games and Apps menu or Java Tools menu).
    - J2ME menu items must be approved by Technical Marketing, CxD and Product Marketing.
    - Position of the “Invisible Net” J2ME menu item(s) in the menu structure must be approved by the appropriate representatives from the product team, the region, the customer, CxD and Technical Marketing.
    - All prompts (including menu prompts and menu list items) and URL addresses must be accessible through the PRI interface.
    - All prompts and URLs must be set with an option to either allow or not allow access to the carriers as determined by individual regions and customers.
    - All J2ME related application menu or context-sensitive menu item prompts must be flexible, easy to change and provision as customization will be necessary by region and customer.
    - All J2ME related application menu or context-sensitive menu item prompts and URL addresses must be accessible through the PRI interface.

- CxD media team will provide or approve any J2ME menu icons or graphics.
- Rendering guidelines for J2ME menu prompt items or J2ME menu list items with associated URLs shall be outlined in CxD NES: SYN2_NES_058 documentation.
- The phone should be flexed to allow URLs within J2ME in one of the following combinations:
  - URLs allowed in the Games and Apps menu
  - URLs allowed in the Java Tools menu only
  - URLs allowed in the BOTH Games and Apps menu AND the Java Tools menu
  - URLs not allowed Games and Apps menu AND URLs not allowed in the Java Tools menu (URLs not allowed in either menu).

J2ME Context-Sensitive Menu Options

The J2ME application is required to embed Invisible Net prompts and URLs within the all J2ME context-sensitive menus.

Prompt Requirements

All prompts referred to in this document must be flexible. The following rules apply:

- Position of J2ME menu list items and context-sensitive menu prompt items must be approved by the appropriate representatives from the product team, the region, the customer, CxD and Technical Marketing.
- All prompts (including menu list items) must be approved via the CR process (asyn_prompts class).

MA-specific/translation information: Prompts as stated above will be flexible and may need translation.

User manual impact: (yes/no): Yes

Hardware Requirements

Product specific restrictions may apply based on memory constraints.
Invisible Net for J2ME functionality is available only on products which have browser capability or products which use HTTP Downloading for J2ME applications.

Interoperability Requirements

SDK requirements and Developer/Style Guide:

- The associated right softkey when a URL will be launched shall be GO TO.

- A developer may embed a URL within a context-sensitive menu.

Because the prompts and URLs in this MRS are variable and flexible and must be approved by the product groups, regions and customers, these items must be validated by as accepted.

Backward Compatibility/Flexing

All J2ME menu or context-sensitive menu item prompts must be flexible, easy to change and provision as customization will be necessary by region and customer.

All J2ME menu or context-sensitive menu item prompts and URL addresses must be accessible through the PRI interface.

The phone shall be flexed to allow URLs within J2ME in one of the following combinations:

- URLs allowed in the Games and Apps menu
- URLs allowed in the Java Tools menu only
- URLs allowed in the BOTH Games and Apps menu AND the Java Tools menu
- URLs not allowed Games and Apps menu A

29
Download Midlet through PC

To download MIDlets through a PC, make a connection to a PC through IrDA, Bluetooth, USB or Serial Cable (RS 232). This content considers only the RS232 connection using JAL.

Establishing Connection

When a successful connection to a PC is made, an application can be downloaded. The MS should display that a connection has been made. Only one connection will be active at a time.

30
Operator Apps Provisioning

The application provisioning feature uses the existing functionality to deliver a trigger pull of Java applications. The following steps are performed to achieve this functionality.

- The operator sets up a URL with the applications that they want to be pushed. Let us assume that this is snake.jar. The operator sets up a WML page with the reference pointing to snake.jad, the corresponding application descriptor file for snake.jar.
- Operator sends an SMSmessage to all the mobiles with the URL for snake.wml embedded inside.
- When the Synergy message center receives the message, it is treated as a browser message and the go to browser option is available.
- The user selects the Go To option to launch a browser that goes to theURL embedded inside the message.
- The browser loads the page setup by the operator. At this point, browse through the page and select the midlets to be downloaded. The download process is described in the chapter 32 download through the browser.

31
MIDP 2.0 Security Model

The following sections describe the MIDP 2.0 Default Security Model for the Motorola C381p handset. The chapter discusses the following topics:

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

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

Refer to the Table 26 for the default security feature/class support for MIDP 2.0:

Feature/Class	Implementation
All methods for the Certificate interface in the javax.microedition.pki package	Supported
All fields, constructors, methods, and inherited methods for the CertificateException class in the javax.microedition.pki package	Supported
MIDlet-Certificate attribute in the JAD	Supported
A MIDlet suite will be authenticated as stated in Trusted MIDletSuites using X.509 of MIDP 2.0 minus all root certificates processes and references	Supported
Verification of RSA-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
Will preload at least one self authorizing Certificates	Supported
All constructors, methods, and inherited methods for the	Supported

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

Table 26 Security feature/class support for MIDP 2.0

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

Untrusted MIDlet Suites

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

The following are conditions of untrusted MIDlet suites:

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

Untrusted Domain

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

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

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

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

Trusted MIDlet Suites

Trusted MIDlet suites are MIDlet suites in which the integrity of the JAR file can be authenticated and trusted by the device, and bound to a protection domain. The Motorola C381p will use x.509PKI for signing and verifying trusted MIDlet suites.

Security for trusted MIDlet suites will utilize protection domains. Protection domains define permissions that will be granted to the MIDlet suite in that particular domain. A MIDlet suite will belong to one protection domain and its defined permissible actions. For implementation on the Motorola C381p, the following protection domains are supported:

- Manufacturer
- Untrusted – all MIDlet suites that are unsigned will belong to this domain.

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

Permission Types concerning the Handset

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

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

User Permission Interaction Mode

User permission for the Motorola C381p handsets is designed to allow the user the ability to either deny or grant access to the protected API or function using the following interaction modes (bolded term(s) is prompt displayed to the user):

- blanket – grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is uninstalled or the permission is changed by the user. (Never Ask)
- session – grants access to the protected API or function every time it is required by the MIDlet suite until the MIDlet suite is terminated. This mode will prompt the user on or before the final invocation of the protected API or function. (Ask Once Per App Running)
- oneshot – will prompt the user each time the protected API or function is requested by the MIDlet suite. (Always Ask)

- No – will not allow the MIDlet suite access to the requested API or function that is protected. (No Access)

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

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

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

Implementation based on Recommended Security Policy

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

Trusted 3rd Party Domain

A trusted third party protection domain root certificate is used to verify third party MIDlet suites. These root certificates will be mapped to a location on the handset that cannot be modified by the user.

The Table 27 shows the specific wording to be used in the first line of the above prompt:

Protected Functionality	Top Line of Prompt
Data network	"Send Data?"
Data network (server mode)	"Receive Data?"
Comm	"Connect?"
Push	"Auto Start-Up?"
SMS	"Use SMS?"
SMS send	"Send SMS?"
SMS receive	"Receive SMS?"
Access phonebook	"Use Phonebook?"

Dial a call	“Make Phone Call?”
CBS	“Use CBS?”
Receive CBS	“Receive CBS?”

Table 27 Protected Functionality fot top line of prompt

The radio button messages will appear as follows and mapped to the permission types as shown in the Table 28:

MIDP 2.0 Permission Types	Dialog Prompts
Blanket	“Always yes. Do not ask again.”
Session	“Yes, this is running.”
Oneshot	“Only this operation. Ask me again.”
No access	“Not this operation. Ask me again.”
	“Not this running.”
	“No, always denied. Do not ask again.”

Table 28 Dialog Prompts for MIDP 2.0 Permission Types

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

Trusted MIDlet Suites Using x.509 PKI

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

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

Signing a MIDlet Suite

The default security model involves the MIDlet suite, the signer, and public key certificates. A set of root certificates are used to verify certificates generated by the signer. Specially designed certificates for code signing can be obtained from the manufacturer,

operator, or certificate authority. Only root certificates stored on the handset will be supported by the Motorola C381p handset.

Signer of MIDlet Suites

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

MIDlet Attributes Used in Signing MIDlet Suites

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

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

Methods:

1. MIDlet.getAppProperty will return the attribute value from the manifest (JAR) if one id defined. If an attribute value is not defined, the attribute value will return from the application descriptor (JAD) if present.

Creating the Signing Certificate

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

Inserting Certificates into JAD

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

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

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

Note the following:

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

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

Creating the RSA SHA-1 signature of the JAR

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

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

Authenticating a MIDlet Suite

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

Verifying the Signer Certificate

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

1. Get the certification path for the signer certificate from the JAD attributes MIDlet-Certificate-1<m>, where <m> starts a 1 and is incremented by 1 until there is no

attribute with this name. The value of each attribute is abase64 encoded certificate that will need to be decoded and parsed.

2. Validate the certification path using the basic validation process as described in RFC2459 using the protection domains as the source of the protection domain root certificates.
3. Bind the MIDlet suite to the corresponding protection domain that contains the protection domain root certificate that validated the first chain from signer to root.
4. Begin installation of MIDlet suite.
5. If attribute MIDlet-Certificate-<n>-<m> with <n> being greater than 1 are present and full certification path could not be established after verifying MIDlet-Certificate-<1>-<m> certificates, then repeat step 1 through 3 for the value <n> greater by 1 than the previous value.

The Table 29 describes actions performed upon completion of signer certificate verification:

Result	Action
Attempted to validate <n> paths. No public keys of the issuer for the certificate can be found, or none of the certificate paths can be validated.	Authentication fails, JAR installation is not allowed.
More than one full certification path is established and validated.	Implementation proceeds with the signature verification using the first successfully verified certificate path for authentication and authorization.
Only one certification path established and validated.	Implementation proceeds with the signature verification.

Table 29 Actions performed upon completion of signer certificate verification

Verifying the MIDlet Suite JAR

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

1. Get the public key from the verified signer certificate.
2. Get the MIDlet-JAR-RSA-SHA1 attribute from the JAD.
3. Decode the attribute value from base64 yielding a PKCS #1 signature, and refer to RFC 2437 for more detail.
4. Use the signer's public key, signature, and SHA-1 digest of JAR to verify the signature. If the signature verification fails, reject the JAD and MIDlet suite. The MIDlet suite will not be installed or allow MIDlets from the MIDlet suite to be invoked as shown in the Table 30.
5. Once the certificate, signature, and JAR have been verified, the MIDlet suite is known to be trusted and will be installed (authentication process will be performed during installation).

The Table 30 is a summary of MIDlet suite verification including dialog prompts:

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

Table 30 MIDlet suite verification

Appendix A: Audio Mix Table

	Single tons	Tone-Sequence	Wav	MP3	AMR	MIDI	Video w/audio	Video w/o audio
Single tone	No	No	No	No	No	Yes	No	Yes
Tone-sequence	No	No	No	No	No	Yes	No	Yes
WAV	No	No	Partly yes	No	No	Yes	No	Yes
MP3	No	No	No	No	No	Yes	No	Yes
AMR	No	No	No	No	No	Yes	No	No
MIDI	Yes	Yes	Yes	Yes	Yes	No	No	Yes
Video w/audio	No	No	No	No	No	No	No	No
Video w/o audio	Yes	Yes	Yes	Yes	No	Yes	No	no

Table 31 Audio Mix

NOTE: If two wave plays have the same frequency, they can mix

Appendix B: Key Mapping

Key Mapping for the C381p

The table below identifies key names and corresponding Java assignments. All other keys are not processed by Java.

Key	Assignment
0	NUM0
1	NUM1
2	NUM2
3	NUM3
4	NUM4
5	SELECT, followed by NUM5
6	NUM6
7	NUM7
8	NUM8
9	NUM9
STAR (*)	ASTERISK
POUND (#)	POUND
JOYSTICK LEFT	LEFT
JOYSTICK RIGHT	RIGHT
JOYSTICK UP	UP
JOYSTICK DOWN	DOWN
SCROLL UP	UP
SCROLL DOWN	DOWN
SOFTKEY 1	SOFT1
SOFTKEY 2	SOFT2
MENU	SOFT3 (MENU)
SEND	SELECT Also, call placed if pressed on lcdui.TextField or lcdui.TextBox with PHONENUMBER constraint set.
CENTER SELECT	SELECT
END	Handled according to Motorola specification: Pause/End/Resume/Background menu invoked.

Appendix B:
Key Mapping

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

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

Appendix C: Memory Management Calculation

Available Memory

The available memory on the Motorola C381p handset is the following:

- Shared memory for MIDlet storage and removable memory
- 800k Bytes Heap size

Appendix D: FAQ

Online FAQ

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

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

http://www.motocoder.com

Appendix E: HTTP Range

Graphic Description

The following is a graphic description of HTTP Range:

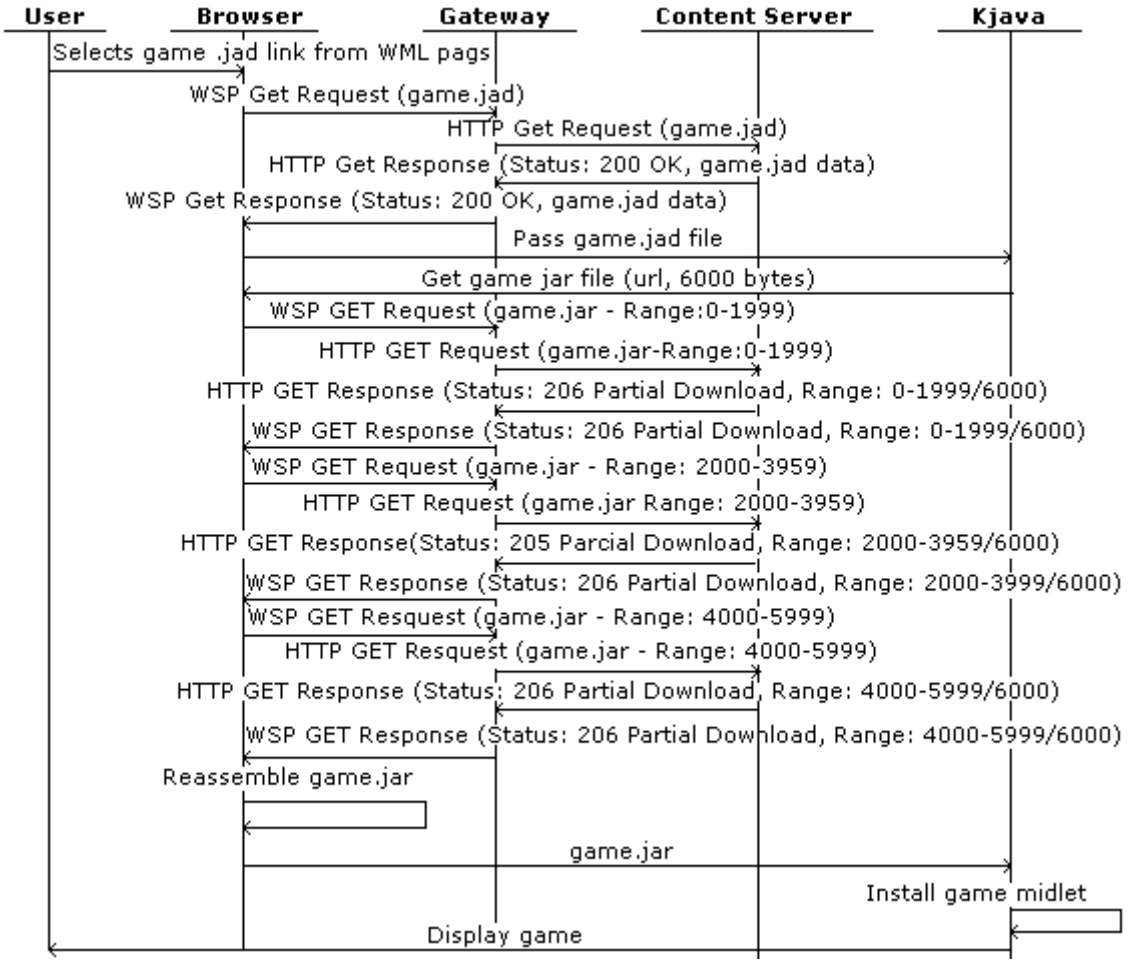


Figure 16 Description of HTTP Range

Appendix F: Spec Sheet

C381p Spec Sheet

Listed below is the spec sheet for the Motorola C381p handset. The spec sheet contains information regarding the following areas:

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

Appendix F:
Spec Sheet

MOTOROLA
intelligence everywhere™

Motorola C381p

Developer Reference Sheet

Technical Specifications

Band/Frequency	GSM 850/900/1800/1900 GPRS
Region	Global
Technology	WAP 2.0, J2ME, SMS, EMS, MMS, AOL/OICQ IM
Connectivity	Mini SB
Dimensions	83.5 x 44 x 22.2
Weight	97 g
Display	Internal: 128 x 128
Operating System	Motorola
Chipset	i250S1

Key Features

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

J2ME™ Information

CLDC v1.0 and MIDP v2.0/v1.0 compliant	
Heap size	800 Kb
Maximum record store size	64 K
MIDlet storage/Shared User Mem	1.5 MB
Interface connections	HTTP, Socket, UDP, Serial port
Maximum number of sockets	4
Supported image formats	PNG, JPEG
Double buffering	Yes
Encoding schemes	ISO8859_1, ISO10646
Input methods	Multitap, iTAP
Additional API's	JSR 120, JSR 135, Moto LWT, Phonebook, Telephone API
Audio	WAV, MIDI, MP3
Video	No
Photo Capture	No

Related Information

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

Tools:
J2ME™ SDK version v4.0
Motorola Messaging Suite v1.1

Documentation:
Creating Media for the Motorola C381p Handset

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

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

Appendix F:
Spec Sheet