Technical Manual

MOTOROLA

Developer Guide

C975 Java ME Developer Guide

Version 02.00

Table of Contents

Index of Figures

Index of Tables

Index of Code Samples

1 Introduction

1.1 Purpose

This document describes the application program interfaces used to develop Motorola compliant Java Platform, Micro Edition (Java ME) applications for the C975 handset supporting CLDC 1.1.

For more detailed information see *Section 3.1.1*.

1.2 Audience

This document is intended for general Java ME developers involved in the production of Java ME applications for the C975 handset.

1.3 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 in regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by

applicable law. Otherwise, they are specifically excluded.

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

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

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

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

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

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

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

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

1.4 References

Reference	Link
Borland	*http://www.borland.com/*
GSM 03.38 standard	*http://www.etsi.org*
GSM 03.40 standard	*http://www.etsi.org*
IBM	*http://www.ibm.com/*
JSR	*http://www.jcp.org*
Motorola	*http://www.motorola.com/*
RFC 2068	*http://www.ietf.org/rfc/rfc2068.txt*
RFC 2396	*http://www.ietf.org/rfc/rfc2396.txt*
SAR	*http://www.wapforum.org*
Sun Java ME	*http://java.sun.com/javame/*
Sun Microsystems	*http://www.sun.com/*
Sun MIDP 2.0 SDK	*http://java.sun.com/products/midp/*

Table 1 References

1.5 Revision History

Version	Date	Reason
00.01	July 08th, 2004	Initial Draft
00.02	August 30th, 2004	Updates after Motorola's review
00.03	September 08th, 2004	Updates after Motorola's review
01.01	June 16th, 2006	Replace J2ME and Motocoder References
01.01	June 16th, 2006	Replace J2ME and Motocoder References
01.02	July 13th, 2006	Remove chipset information on specsheet
02.00	July 13th, 2006	Document approval.

Table 2 Revision History

1.6 Definitions, Abbreviations,

Acronyms

Acronym	Description
AMS	Application Management Software
API	Application Program Interface
BMP	Windows BitMap Format (image extension '.bmp')
CLDC	Connected Limited Device Configuration
DNS	Domain Name System
DRM	Digital Rights Management
GIF	Graphics Interchange Format (image extension '.gif')
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IP	Internet Protocol
IRCOMM	Is a specification from the Infrared Design Association (IRDA) and determines how different devices can talk to each other via infrared.
IrDA	Infrared Data Association
ITU	International Telecommunication Union
JAD	Java Application Descriptor
JAM	Java Application Manager
JAR	Java Archieve. Used by Java ME applications for compression and packaging.
Java ME	Java Platform, Micro Edition (Java ME, formerly J2ME)
JPG	Joint Photographic Experts Group (image extension '.jpg')
JSR	Java Specification Request
JSR-139	Java Specification Request 139. Defines a revised version of the Java ME Connected, Limited Device Configuration (CLDC)
JVM	Java Virtual Machine
KVM	K Virtual Machine (Java ME runtime environment)
MIB	Motorola Internet Browser
MIDP	Mobile Information Device Profile
MMA	Multimedia API
MT	Mobile Terminated
OEM	Original Equipment Manufacturer
P2K	Motorola Plataform 2000
PNG	Portable Network Graphics (image extension '.png')
RFC	Request for Comments

RMS	Record Management System
SD/MMC	Secure Digital Card / Multi Media Card
SDK	Software Development Kit
SMS	Short Message Service
SMSC	Short Messaging Service Center
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
Trusted Device	A paired device that is explicitly marked as trusted.
UDP	User Datagram Protocol
UI	User Interface
URI	Unified Resource Identifier
URL	Universal Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine
WMA	Wireless Messaging API

Table 3 Definitions, Abbreviations, Acronyms

1.7 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 - Java ME Introduction: This chapter describes the Java ME platform and the available resources on this Handset.
Chapter 3 - Developing and Packaging Java ME Applications: This chapter describes important features to look for selecting tools and emulation environments. It also describes how to package a Java ME 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 - Shared JAD URLs: This chapter describes the Share JAD URLs, it allows users to share their downloaded Java ME application URLs with others.
Chapter 7 - JAD Attributes: This chapter describes what attributes are supported.
Chapter 8 - iTAP: This chapter describes iTAP support.
Chapter 9 - Network APIs: This chapter describes the Java Networking API

and network access.
Chapter 10 - CommConnection Interface: This chapter describes the CommConnection API.
Chapter 11 - MIDP 2.0 Security Model: This chapter describes the MIDP 2.0 default security model."
Chapter 12 - JSR-120 - Wireless Messaging API: This chapter describes JSR-120 implementation.
Chapter 13 - JSR-135 - Mobile Media API: This chapter describes image types and supported formats.
Chapter 14 - JSR-139 - CLDC 1.1: This chapter describes briefly some characteristics of CLDC 1.1 and presents additional classes, fields, and methods supported for CLDC 1.1.
Chapter 15 - MIDlet storage in removable memory: This chapter details storage, installation and access of Java ME applications in removable memory.
Chapter 16 - JSR-185 - JTWI: This chapter describes JTWI functionality.
Chapter 17 - JSR-184 - Mobile 3D Graphics API: This chapter describes the JSR-184 which defines an API for rendering three-dimensional (3D) graphics.
Appendix A - Key Mapping: This appendix describes the key mapping of the Motorola C975 handset, including the key name, key code, and game action of all Motorola keys
Appendix B - Memory Management Calculation: This chapter describes the memory management calculations.
Appendix C - FAQ: This appendix provides a link to the dynamic online FAQ.
Appendix F - Spec Sheet: This appendix provides the spec sheet for the Motorola C975 handset.
Appendix H - Quick Reference: This appendix provides quick references to this document.

2 Java ME Introduction

The Motorola C975 handset includes the Java Platform, Micro Edition, also known as the Java ME platform. The Java ME platform enables developers to easily create a variety of Java applications ranging from business applications to games. Prior to its inclusion, services or applications residing on small consumer devices like cell phones could not be upgraded or added to without significant effort. By implementing the Java ME platform on devices like the Motorola C975 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 Java ME environment and the tools that can be used to develop applications for the Motorola C975 handset.

2.1 The Java Platform, Micro Edition (Java ME)

The Java ME 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 Java ME architecture, see Figure 1 , 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.

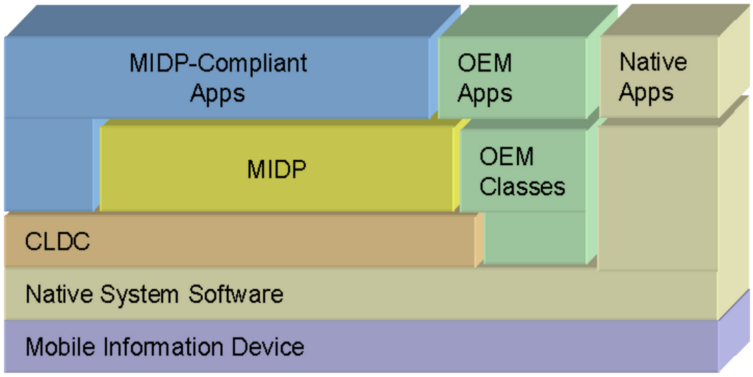

Figure 1 Java ME Architecture

The Configuration Layer used in the Motorola C975 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 Java ME, see the Sun Java ME documentation (*http://java.sun.com/javame/*).

2.2 The Motorola Java ME Platform

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

The Motorola C975 handset contains OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the Motorola C975 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

2.3 Resources and APIs Available

MIDP 2.0 will provide support to the following functional areas on the Motorola C975

handset:

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

Additional Functionality

- JSR-118
- JSR-120
- JSR-135
- JSR-139
- JSR-184
- JSR-185

3 Developing and Packaging Java ME Applications

3.1 Guide to Development in Java ME

Introduction to Development

This appendix assumes the reader has previous experience in Java ME 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 Java ME handsets.

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

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

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

The MIDlet will consist of two core specifications, namely Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). Both of these specifications (JSR - 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.

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

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

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

In addition to the IDEs and Sun SDK for development, Motorola offers access to our own SDK 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 MOTODEV website.

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

4 Downloading Applications

4.1 Methods of Downloading

The load of applications (MIDlets) in Motorola devices that consist of the transmission of an application from PC to device can be carried through the direct cable USB, via CE Bus.

The direct cable approach can be performed using a tool available from MOTODEV called MIDway. The version available of writing is , which supports USB cable download.

It is important to note that the MIDway tool will only work with a device that has been enabled to support direct cable Java download. This feature is not available by purchasing a device through a standard consumer outlet.

The easiest method of confirming support for this is by looking at the "Java Tool" menu on the phone in question and seeing if a "Java app loader" option is available on that menu. If it is not, then contact MOTODEV support for advice on how to receive an enabled handset.

For more information about MIDway tool can be obtained through the MOTODEV website (*http://developer.motorola.com*).

5 Application Management

The following sections describe the application management scheme for the Motorola C975 handset. This chapter will discuss the following:

- Downloading a JAR without a JAD
- Installation and Deletion Status Reports
- DRM Content Protection in Java

5.1 Downloading a JAR file without a JAD

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

5.2 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 al-

lowed to use it. In some instances, if the status report cannot be sent, the MIDlet will be deleted by operator's request. 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.

The following 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

5.3 DRM Content Protection in Java

Digital Rights Management (DRM) is a method to prevent MIDlets from distributing DRM content using any packet data network connection. In others words, DRM is a method of protecting content from illegal distribution by embedding the content into an encrypted package, along with rules dictating its use.

If the user has a set of keys and a valid license, then they are used for a specific file. A DRM application is required to decrypt the content for playback. This method will be transparent for the user, if he has a valid license.

The invalid license might happen because elapsed number of times the content to be executed/played, or elapsed validity for the license, or the content received through separate delivery.

For more information about this method, see at *http://www.openmobilealliance.org*.

6
Shared JAD URLs

6.1 Overview

Actually, users are able to download Java ME 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 Java ME application URLs with others. When Java ME applications are downloaded, the browser shall provide the Java Application Manager (JAM) with the JAD URL address. When Java ME applications are downloaded via PC or MMS, a new JAD attribute shall specify the JAD URL address.

6.2 Tell-A-Friend Option

When entering the Java ME 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.

7 JAD Attributes

7.1 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. Table 4 lists all MIDlet attributes, descriptions, and its location in the JAD and/or JAR manifest that are supported in the Motorola implementation. Please note that the MIDlet will not install if the MIDlet-Data-Size is greater than 512k.

Attribute Name	Attribute Description	JAR Manifest	JAD
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlets to the user	Yes	Yes

MIDlet-Version	The version number of the MIDlet suite	Yes	Yes
MIDlet-Vendor	The organization that provides the MIDlet suite.	Yes	Yes
MIDlet-Icon	The case-sensitive absolute name of a PNG file within the JAR used to represent the MIDlet suite.	Yes	Yes
MIDlet-Description	The description of the MIDlet suite.	No	No
MIDlet-Info-URL	A URL for information further describing the MIDlet suite.	Yes	No
MIDlet-<n>	The name, icon, and class of the 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.	Yes	Yes
MicroEdition-Profile	The Java ME 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 Java ME Configuration required, i.e CLDC	Yes, or no if included in the JAD.	Yes, or no if included in the JAR Manifest.
MIDlet-Permissions	Zero or more permissions that are critical to the function of the MIDlet suite.	Yes	Yes
MIDlet-Permissions-Opt	Zero or more permissions that are non-critical to the function of the MIDlet suite.	Yes	Yes
MIDlet-Push-<n>	Register a MIDlet to handle inbound connections	Yes	Yes

MIDlet-Install-Notify	The URL to which a POST request is sent to report installation status of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Notify	The URL to which a POST request is sent to report deletion of the MIDlet suite.	Yes	Yes
MIDlet-Delete-Confirm	A text message to be provided to the user when prompted to confirm deletion of the MIDlet suite.	Yes	Yes
FlipInsensitive	MIDlets with this Motorola specific attribute will enable the MIDlet to run with the flip closed.	Yes	Yes
Background	MIDlets with this Motorola specific attribute will continue to run when not in focus.	Yes	Yes

Table 4 MIDlet attributes, descriptions, and its location in the JAD and/or JAR manifest

8
iTAP

8.1 Intelligent Keypad Text Entry API

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

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

Predictive text entry allows a user to simply type in the letters of a word using only one key press per letter, as 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 Java ME 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 5 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 5 iTAP feature/class

9 Network APIs

9.1 Network Connections

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

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

Refer to Table 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	
HttpConnection interface in the javax.microedition.io package	Supported
HttpsConnection interface in the javax.microedition.io package	Supported
SecureConnection interface in the javax.microedition.io package	Supported
SecurityInfo interface in the javax.microedition.io package	Supported
UDPDDatagramConnection interface in the	Supported

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

Table 6 Network API feature/class support for MIDP

Code Sample 1 shows the implementation of Socket Connection:

Socket Connection

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

...

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

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

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

        //read data from server until done
            do
            {
                bytes_read = is.read(buffer, offset, bytes_left);

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

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

Code Sample 1 Socket Connection

9.2 User Permission

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

9.3 Indicating a Connection to the User

When the java implementation makes any of the additional network connections, it will indicate to the user that the handset is actively interacting with the network. To

indicate this connection, the network icon will appear on the handset's status bar as shown in Figure 2 .

Figure 2 Network Connections example

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

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

9.4 HTTPS Connection

Motorola implementation supports a HTTPS connection on the Motorola C975 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*

Code Sample 2 shows the implementation of HTTPS:

HTTPS

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

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

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

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

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

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

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

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

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

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

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

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

Code Sample 2 HTTPS

9.5 DNS IP

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

9.6 Push Registry

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

9.7 Mechanisms for Push

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

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

the push notification.

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

9.8 Push Registry Declaration

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

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

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

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

Dynamically register using an assigned port number

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

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

Code Sample 3 shows the implementation of Push Registry:

Push Registry Declaration

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

```
import javax.microedition.io.PushRegistry;

public class PushTest_1 extends MIDlet implements CommandListener{

        public Display display;

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

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

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

        public static Command mc;
        public static Displayable ms;

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

            display = Display.getDisplay(this);

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

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

            mainForm.append("Press \"Reg\" softkey to register a new connec-
tion.\n" +
                            "Press \"Unreg\" softkey to unregister a connection.");
            mainForm.addCommand(exitCommand);
```

```
        mainForm.addCommand(unregCommand);
        mainForm.addCommand(regCommand);
        mainForm.setCommandListener(this);

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

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

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


    }
    public void pauseApp(){}


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

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

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

    public void commandAction(Command c, Displayable s) {

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

        if((c == regCommand) && (s == mainForm)){
```

```
            display.setCurrent(regForm);
         }

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

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

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

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

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

      }

      public class runThread extends Thread{
         public void run(){
            if((mc == unregCommand) && (ms == mainForm)){
               try{
                  registeredConns = PushRegistry.listConnections(false);
                  if(unregForm.size() > 0) unregForm.delete(0);
                  registeredConnsCG = new ChoiceGroup("Connections",
ChoiceGroup.MULTIPLE, registeredConns, null);
                  if(registeredConnsCG.size() > 0) unreg-
Form.append(registeredConnsCG);
                  else unregForm.append("No registered connections
found.");
                  display.setCurrent(unregForm);
```

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

            }

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

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

WakeUp.java

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

public class WakeUp extends MIDlet implements CommandListener{

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


    public void startApp() {

        display = Display.getDisplay(this);

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



        display.setCurrent(mainForm);

    }

    public void pauseApp() {
    }

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

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

```
        }
        if(c == registerCommand){

            new regThread().start();

        }
    }

    public class regThread extends Thread{


        public void run(){

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

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

                System.out.println(curTime + delay);

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

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

        }
    }
}
```

SMS_send.java

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

public class SMS_send extends MIDlet implements CommandListener{
```

```
    public Display display;

    public static Form messageForm;
    public static Form mainForm;

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

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

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

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

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

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

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


        display = Display.getDisplay(this);

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

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

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

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


    }

    public void pauseApp(){
    }

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

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

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

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

```
    }

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

Code Sample 3 Push Registry

9.9 Delivery of a Push Message

A push message intended for a MIDlet on the Motorola C975 handset will handle the following interactions:

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

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

as shown in Figure 3 .

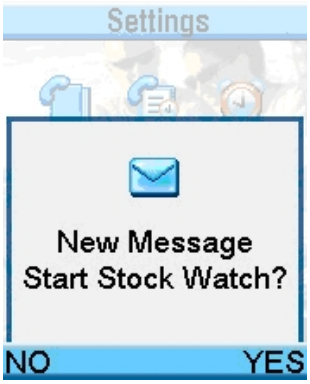

Figure 3 Intend Application Run Option

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

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

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

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

9.10 Deleting an Application Registered for Push

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

9.11 Security for Push Registry

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

9.12 Network Access

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

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

10 CommConnection Interface

10.1 CommConnection

The CommConnection interface defines a logical serial port connection. A logical serial port connection is a logical connection through which bytes are transferred serially. This serial port is defined within the underlying operating system and may not correspond to a physical RS-232 serial port. For example, IrDA IRCOMM ports can be configured as a logical serial port within the operating system so it can act as a logical serial port.

10.2 Accessing

The Comm port is accessed using a Generic Connection Framework string with an explicit port identifier and embedded configuration parameters, each separated with a semi-colon (;). Only one application may be connected to a particular serial port at a given time. A `java.io.IOException` is thrown if an attempt is made to open the serial port with `Connector.open()` if the connection is already open.

A URI with the type and parameters is used to open the connection. The scheme, as defined in RFC 2396, will be the following:

- `Comm.: <port identifier> [<optional parameters>]`

10.3 Parameters

The first parameter will be a port identifier, which is a logical device name. These port identifiers are device specific and should be used with care.

The valid identifiers for a particular device and OS can be queried through the `System.getproperty()` method using the key microedition.commports. A list of ports, separated by commas, is returned which can be combined with a comm: prefix as the URL string to open a serial port connection.device specific and should be used with care.

The valid identifiers for a particular device and OS can be queried through the `System.getproperty()` method using the key `microedition.commports`. A list of ports, separated by commas, is returned which can be combined with a comm: prefix as the URL string to open a serial port connection.

Any additional parameters will be separated by a semi-colon (;) without spaces. If a particular parameter is not applicable to a particular port, the parameter will be ignored. The port identifier cannot contain a semi-colon (;).

Legal parameters are defined by the definition of the parameters below. Illegal or unrecognized parameters cause an `IllegalArgumentException`. If the value of a parameter is supported by the device, it will be honored. If the value of a parameter is not supported, a `java.io.IOException` is thrown. If a baudrate parameter is requested, it is treated the same way that a `setBaudRate` method handles baudrates. For example, if the baudrate requested is not supported, the system will substitute a valid baudrate which can be discovered using the `getBaudRate` method.

The Table 7 describes optional parameters.

Parameter	Default	Description
baudrate	platform dependent	The speed of the port.
bitsperchar	8	The number bits per character(7 or 8).
stopbits	1	The number of stop bits per char(1 or 2)
parity	none	The parity can be odd, even, or none.

blocking	on	If on, wait for a full buffer when reading.
autocts	on	If on, wait for the CTS line to be on before writing.
autorts	on	If on, turn on the RTS line when the input buffer is not full. If off, the RTS line is always on.

Table 7 Interface Commconncetion optional parameters

10.4 BNF Format for Connector.open () string

The URI must conform to the BNF syntax specified in Table 8 . If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

BNF syntax	
<comm_connection_string>	::= "**comm:**"<port_id>[<options_list>] ;
<port_id>	::= *string of alphanumeric characters*
<options_list>	::= *(<baud_rate_string>| <bitsperchar>| <stopbits>| <parity>| <blocking>| <autocts>| <autorts>) ; ; if an option duplicates a previous option in the ; option list, that option overrides the previous ; option
<baud_rate_string>	::= ";**baudrate**="<vbaud_rate>
<baud_rate>	::= *string of digits*
<bitsperchar>	::= ";**bitsperchar**="<bit_value>
<bit_value>	::= "7" | "8"
<stopbits>	::= ";**stopbits**="<stop_value>
<stop_value>	::= "1" | "2"
<parity>	::= ";**parity**="<parity_value>
<parity_value>	::= "even" | "odd" | "none"
<blocking>	::= ";**blocking**="<on_off>
<autocts>	::= ";**autocts**="<on_off>
<autorts>	::= ";**autorts**="<on_off>
<on_off>	::= "on" | "off"

Table 8 Interface Commconncetion BNF syntax

10.5 Comm Security

Access to serial ports is restricted to prevent unauthorized transmission or reception of data. The security model applied to the serial port connection is defined in the implementing profile. The security model will be applied on the invocation of the `Connector.open ()` method with a valid serial port connection string. Should the application not be granted access to the serial port through the profile authorization scheme, a `java.lang.SecurityException` will be thrown from the `Connector.open ()` method. The security model will be applied during execution, specifically when the methods `openInputStream(), openDataInputStream(), openOutputStream(), and openDataOutputStream()` are invoked.

The Code Sample 4 shows the implementation of CommConnection:

Sample of a CommConnection accessing a simple loopback program

```
CommConnection cc = (CommConnection)
          Connector.open("comm:com0;baudrate=19200");
 int baudrate = cc.getBaudRate();
 InputStream is  = cc.openInputStream();
 OutputStream os = cc.openOutputStream();
 int ch = 0;
 while(ch != 'Z') {
    os.write(ch);
    ch = is.read();
    ch++;
 }
 is.close();
 os.close();
 cc.close();
```

Sample of a CommConnection discovering available comm Ports

```
String port1;
 String ports = System.getProperty("microedition.commports");
 int comma = ports.indexOf(',');
 if (comma > 0) {
    // Parse the first port from the available ports list.
    port1 = ports.substring(0, comma);
 } else {
    // Only one serial port available.
```

```
   port1 =ports;
}
```

Code Sample 4 CommConnection implementation

10.6 Port Naming Convention

Logical port names can be defined to match platform naming conventions using any combination of alphanumeric characters. Ports will be named consistently among the implementations of this class according to a proposed convention. VM implementations will follow the following convention:

- Port names contain a text abbreviation indicating port capabilities followed by a sequential number for the port. The following device name types will be used:
  - COM# - COM is for RS-232 ports and # is a number assigned to the port
  - IR# - IR is for IrDA IRCOMM ports and # is a number assigned to the port

The naming scheme allows API users to determine the type of port to use. For example, if an application "beams" a piece of data, the application will look for IR# ports for opening the connection.

10.7 Method Summary

The Table 9 describe the CommConnection method summary for MIDP .

Method Summary	
int	`getBaudRate()` Gets the baudrate for the serial port connection
int	`setBaudRate (int baudrate)` Sets the baudrate for the serial port connection

Table 9 Method Summary

11 MIDP 2.0 Security Model

The following sections describe the MIDP 2.0 Default Security Model for the Motorola C975 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 Motodev website (*http://developer.motorola.com*).

Refer to Table 10 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
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 SHA-1 signatures with a MD5 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	Supported

javax.microedition.pki package	
Will preload two self authorizing Certificates	Supported
All constructors, methods, and inherited methods for the MIDletStateChangeException class in the javax.microedition.midlet package	Supported
All constructors and inherited methods for the MIDletStateChangeException class in the javax.microedition.midlet package	Supported

Table 10 MIDP 2.0 Feature/Class

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

11.1 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 one or more 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.

11.2 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

11.3 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 C975 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 C975, the following protection domains should exist:

- Manufacturer - permissions will be marked as "Allowed" (Full Access). Downloaded and authenticated manufacturer MIDlet suites will perform consistently with MIDlet suites pre-installed by the manufacturer.
- Operator - permissions will be marked as "Allowed" (Full Access). Downloaded and authenticated operator MIDlet suites will perform consistently with other MIDlet suites installed by the operator.
- 3rd Party - permissions will be marked as "User". User interaction is required for permission to be granted. MIDlets do not need to be aware of the security policy except for security exceptions that will occur when accessing APIs.
- 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.

11.4 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 permission 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 permission that requires 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.

11.5 User Permission Interaction Mode

User permission for the Motorola C975 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 the 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**)
- 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.

11.6 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.

11.7 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 11 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?
Record audio/video	Record?
Capture snapshot image	Video capture?
Access File System	Using File?

Table 11 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 12 :

MIDP 2.0 Permission Types	Dialogs 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 12 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.

11.8 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).

11.9 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 C975 handset.

11.10 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).

11.11 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 HAD 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.

11.12 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.

11.13 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.

11.14 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 for 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.

11.15 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.

11.16 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 at 1 and is incremented by 1 until there is no attribute with this name. The value of each attribute is a base64 encoded certificate that 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> is 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 13 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 13 Actions performed of signer certificate verification

11.17 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 13

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).

Table 14 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 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 14 Summary of MIDlet suite verification

11.18 Bound Certificates

Bound certificates enable an efficient process to aid developers in the MIDlet development and testing phase when working with signed applications. Currently the delay for the developer occurs because specific flex files need to be created for each developer and for each domain being tested.

12 JSR-120 - Wireless Messaging API

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

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

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

12.2 SMS Client Mode and Server Mode Connection

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

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

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

Messages received with this identifier 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");
```

12.3 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 Java ME 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 sections A.4.0 and A.6.0 of the JSR-120.

When a MIDlet in server mode requests a port number (identifier) to use and it is the first MIDlet to request this identifier it 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, 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.

12.4 SMS Storing and Deleting Received Messages

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

12.5 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.

12.6 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.

12.7 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 Java ME MIDlet. This only applies to trusted, signed MIDlets.
- If a message is received and the untrusted unsigned application and the KVM are not running then the message will be discarded.
- There is a SMS Access setting in the Java Settings menu option on the handset that allows the user to specify when and how often to ask for authorization. Before the connection is made from the MIDlet, the options available are:

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

The Table 15 is a 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
Removal of SMS messages	Supported
Terminated SMS removal - any user prompts handled by MIDlet	Supported
Originated SMS removal - any user prompts handled by MIDlet	Supported
All fields, methods, and inherited methods for the Connector Class in the javax.microedition.io package	Supported
All methods for the BinaryMessage interface in the javax.wireless.messaging package	Supported
All methods for the Message interface in the javax.wireless.messaging package	Supported
All fields, methods, and inherited methods for the MessageConnection interface in the javax.wireless.messaging package	Supported
Number of MessageConnection instances in the javax.wireless.messaging package	32 maximum
Number of MessageConnection instances in the javax.wireless.messaging package	16
All methods for the MessageListener interface in the javax.wireless.messaging package	Supported
All methods and inherited methods for the TextMessage interface in the javax.wireless.messaging package	Supported
16 bit reference number in concatenated messages	Supported
Number of concatenated messages.	30 messages in inbox, each can be concatenated from 3 parts. No limitation on outbox (immediately transmitted)
Allow MIDlets to obtain the SMSC address with the wireless.messaging.sms.smsc system property	Supported

Table 15 List of Messaging features/classes

The Code Sample 5 shows 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://" + outAd-
dr);

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

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

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

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

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

Creation of server connection:

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

Creation of client connection with port number:

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

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

Getting of address:

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

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

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

Getting of timestamp for the message:

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

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://" + outAd-
dr);

      /* Create BinaryMessage for client mode */
      binMsg =
(BinaryMessage)connClient.newMessage(MessageConnection. BIN-
ARY_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 WMA

12.8 App Inbox Clean-up

Actually, messages for MIDlets are stored in a separate App Inbox. This App Inbox is cleaned up automatically.

The App Inbox capacity is 26 messages or 26 segments and when a new message is received for a certain port number, and the App Inbox capacity has reached its limit of 26 messages, then the messages in the App Inbox will be deleted in the following order:

- If a certain port number has any unread messages in the App Inbox, then the oldest unread message in the buffer relative to that port number WILL be deleted next.
- If a certain port number currently has no messages in the App Inbox, then the oldest unread message in the buffer relative to all port numbers will be deleted next.

When the maximum number of messages is reached and the phone has reached memory full condition, no new messages can be received by the applications. A blinking messaging icon is used to inform the user that the messaging folder is full. At this stage the user has to manually delete some messages to clear some memory to allow the reception of incoming messages.

13 JSR-135 - Mobile Media API

13.1 Network Connections

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

The new implementation of JSR-135 supports the playback of more audio formats and recording of time-based media-audio and video as well as still-image capture.

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 6 to show 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/mp3");
}
catch (Exception e)
```

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

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

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

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

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

// Release the resources
```

```
player.close();
```

Code Sample 6 JSR-135 MMA

13.2 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.

13.3 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. Depending on the application, this will need to be mapped to the volume level on the phone (0-7).
- Specifying Volume in the Level Scale - specifies volume in a linear scale. It ranges from 0 - 100 where 0 represents silence and 100 represents the highest volume available.
- Mute - setting mute on or off does not change the volume level returned by the getLevel. 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.

13.4 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.

13.5 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.

13.6 Audio Media

Table 16 describes multimedia file formats are supported:

File Type	CODEC
WAV	PCM
WAV	ADPCM

MP3	MPEG-1 layer III
SP MIDI	General MIDI
MIDI Type 0	General MIDI
MIDI Type 1	General MIDI
BAS	General MIDI

Table 16 Multimedia file formats

Table 17 is a list of audio MIME types supported:

Category	Description	MIME Type
Audio	MIDI	audio/midi x-midi mid x-mid sp-midi
Audio	MP3 Audio	audio/mpeg
Audio	WAV	audio/wav x-wav
Audio	AMR	audio/amr audio/mp4
Audio	iMelody	audio/imy

Table 17 Audio MIME types

Table 18 decepts multimedia feature/class support for JSR-135:

Feature/Class	Implementation
Media package found	Supported
Media control package	Supported
Media Protocol package	Streaming not 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	Supported

interface in javax.microedition.media.control	
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
Multiple SourceStreams for a DataSource	2

Table 18 Multimedia feature/class support for JSR-135

NOTE: The multimedia engine only supports prefetching 1 sound at a time, but 2 exceptions exist where 2 sounds can be prefetched at once. These exceptions are listed below:

1. Motorola provides the ability to play MIDI and WAV files simultaneously, but the MIDI track must be started first. The WAV file should have the following format: PCM 8,000 Khz; 8 Bit; Mono
2. When midi, iMelody, mix, and basetracks are involved, two instances of midi, iMelody, mix, or basetrack sessions can be prefetched at a time, although one of these instances has to be stopped. This is a strict requirement as (for example) two midi sounds cannot be played simultaneously.

13.7 Mobile Media Feature Sets

Table 19 lists the packages, classes, fields, and methods that must/should be implemented for Phase II of JSR-135 in addition to the Phase I implementation of JSR-135.

Appropriate exception shall be generated if the called method is not supported by the implementation. If a method is accessed without proper security permissions, security exception shall be thrown.

Package	Classes	Methods	Comments & Requirements
javax.microedition.	TempoCon-	setTempo()	Sets the current

media. control	trol (Applicable to MIDI/iMelody audio formats. Implementa-tion guidance - SHOULD.)		playback tempo. MUST implement a tempo range of 10 to 300 beats per minute.
		getTempo()	Gets the current playback tempo.
	PitchControl Applicable to MIDI / iMelody audio formats. Implementa-tion guidance - SHOULD)	getMax-Pitch()	Gets the maximum playback pitch raise supported by the player. SHOULD implement a maximum playback pitch raise of 12,000 milli-semitones.
		getMinPitch()	Gets the minimum playback pitch raise supported by the player. SHOULD implement a minimum playback pitch raise of 12,000 mil-lisemitones.
		getPitch()	Gets the current playback pitch raise.
		setPitch()	Sets the relative pitch raise.
	FramePosi-tioningC ontrol (Implementa tion guidance - SHOULD)	mapFrameT-oTime()	Converts the given frame number to the corresponding media time.
		mapTimeTo-Frame()	Converts the given media time to the corresponding frame number.
		seek()	Seeks to a given video

			frame.
		skip()	Skips a given number of frames from the current position.
	MIDIControl (Implementation guidance - SHOULD)	All fields & methods	
	RecordControl	All fields & methods	RecordControl controls the recording of media from a Player. Supports all methods. Required for audio capture functionality. Video capture support is optional. RecordControl is a protected API as specified in the Security section.
	VideoControl (Implementation guidance - SHOULD)	All fields & methods. getSnapshot() method MUST be supported if the VideoControl is implemented by an instance of camera device. If VideoControl is implemented by video player for video file/ stream playback, it is not mandatory to support get	VideoControl controls the display of video. A Player which supports the playback of video MUST provide a VideoControl via its getControl and getControls methods.

		Snap Shot() method.	
	MetaData-Control	Implement all fields and methods. Support title, copyright, data, author keys for CODECs supporting these keys.	
javax.microedition.media	Player	All fields and methods	SHOULD allow a Player to use a different TimeBase other than its own. This is required for synchronization between multiple Media Players.
	PlayerListener	All fields and methods	SHOULD let applications register PlayerListener for receiving Player events.
	Manager	All fields and methods	MUST support file locator for local playback. For streaming, RTP locator needs to be supported. For camera, new device locator, "camera" has to be supported.
	TimeBase	getTime()	Gets the current time of this TimeBase.
javax.microedition.media.protocol	content-Descriptor	getcontent-Type()	Obtains a string that represents the content type for this descriptor.

Table 19 Packages, classes, fields, and methods implemented for Phase II of JSR-135

13.8 Audio Mixing

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

13.9 Media Locators

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

13.10 RTP locator

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

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

13.11 HTTP Locator

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

E.g.: Manager.createPlayer("http://webserver/tune.mid")

13.12 File Locator

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

E.g.: Manager.createPlayer("file://motorola/audio/sample.mid")

13.13 Capture Locator

Capture Locator should be supported for audio and video devices. A new device "camera" must be defined and supported for camera device. Manager.createPlayer() call shall return camera player as a special type of video player. Camera player should implement VideoControl and should support taking snapShots using VideoControl.getSnapshot() method. E.g.: Manager.createPlayer("capture://camera")

NOTE: For mandatory capture formats, refer to section 0.0.4. Refer to JSR-135 API for capture locator syntax.

13.14 Supported Multimedia File Types

The following sections have tables that list multimedia file types (with corresponding CODECs) that should be supported in products that are JSR-135 compliant in addition to JSR-135 Mobile API Phase I. The common guideline being all codecs and file types supported by native side should be accessible through JSR-135 implementation. The implementation of JSR-135 (and these tables) must be updated every time a new file type and/or CODEC is released. Multimedia File Type Support.

13.15 Image Media

File Type	CODEC	Functionality
JPEG	JPEG	Capture

Table 20 Image Media

13.16 Audio Media

File Type	CODEC	Functionality
MP3	MPEG-1 layer III	Playback

WAV	PCM	Playback
AMR	AMR NB	Playback and Capture

Table 21 Audio Media

13.17 Video Media

File Type	CODEC	Functionality
MPEG4	MPEG4 with or without AMR audio (NOT Supported for KJAVA Application).	Playback
H.263	H.263 with or without AMR audio (NOT Supported for KJAVA Application).	Playback

Table 22 Video Media

13.18 Security

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

13.19 Policy

Table 23 security policy will be flexed in per operator requirements at ship time of the handset.

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

Table 23 Security policy

13.20 Permissions

Table 24 lists individual permissions within Multimedia Record function group.

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

Table 24 Permissions within Multimedia Record

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

14 JSR-139 - CLDC 1.1

14.1 JSR-139

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

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

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

	Classes	Additional Fields/ Methods	Comments
System Classes	Java.lang.Thread	Thread (Runnable target, String name)	Allocates a new Thread object with the given target and name
		Thread (String name)	Allocates a new Thread object with the given name
		String getName ()	Returns this thread's name
		Void interrupt ()	Interrupts this

			thread
	Java.lang.String	Boolean equalIgnoreCase (String anotherString)	Compares this string to another String, ignoring case considerations
		String intern ()	Returns a canonical representation for the string object
		Static String valueOf (float f)	Returns the string representation of the float argument
		Static String valueOf (double d)	Returns the string representation of the double argument
Data Type Classes	Java.lang.Float		New Class: Refer to CLDC Spec for more details
	Java.lang.Double		New Class: Refer to CLDC Spec for more details
Calendar and Time Classes	Java.util.Calendar	Protected int [ ] fields	The field values for the currently set time for this calendar
		Protected boolean { } is set	The flags which tell if a specified time field for the calendar is set
		Protected long time	The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT
		Protected abstract void ComputeFields	Converts the current millisecond time value to field values in fields [ ]
		Protected abstract	Converts the cur-

		void ComputeTime	rent field values in fields [ ] to the millisecond time value time
	Java.lang.Date	String toString ()	Converts this date object to a String of the form: Dow mon dd hh:mm:ss zzz yyyy
Exception and Error Classes	Java.lang.NoClassDefFoundError		New Class: Refer to CLDC Spec for more details
Weak References	Java.lang.ref.Reference		New Class: Refer to CLDC Spec for more details
	Java.lang.ref.WeakReference		New Class: Refer to CLDC Spec for more details
Additional Utility Classes	Java.util.Random	Double nextDouble ()	Returns the next-pseudorandom, uniformly distributed double value between 0.0 and 1.0 from the random number generator's sequence
		Float nextFloat ()	Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from the random number generator's sequence
		Int nextInt (int n)	Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value

			(exclusive), drawn from this random number generator's sequence
	Java.lang.Math	Static double E	The double value that is closer than any other to e, the base of the natural logarithms
		Static double PI	The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter
		Static double abs (double a)	Returns the absolute value of a double value
		Static float abs (float a)	Returns the absolute value of a double value
		Static double ceil (double a)	Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer
		Static double cos (double a)	Returns the trigonometric cosine of an angle
		Static double floor (double a)	Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.

		Static double max (double a, double b)	Returns the greater of two double values
		Static float max (float a, float b)	Returns the greater of two float values
		Static double min (float a, float b)	Returns the smaller of two double values
		Static float min (float a, float b)	Returns the smaller of two float values
		Static double sin (double a)	Returns the trigonometric sine of an angle
		Static double sqrt (double a)	Returns the correctly rounded positive square root of a double value
		Static double tan (double a)	Returns the trigonometric tangent of angle
		Static double todegrees (double angrad)	Converts an angle measured in radians to the equivalent angle measured in degrees
		Static double toradians (double angdeg)	Converts an angle measured in degrees to the equivalent angle measured in radians

Table 25 Additional classes, fields, and methods supported for CLDC 1.1 compliance

15 MIDlet storage in removable memory

15.1 Overview

Motorola Java ME enabled devices with removable memory like SD/MMC cards will be able to store, install and access Java ME Applications on removable memory or phone memory once the application gets downloaded. The user also can to list and launch Java ME Applications stored on removable memory.

Other feature included is a mechanism to DRM protects Java ME Applications installed in secondary memory.

15.2 Installing downloaded applications into removable memory

The Java ME Application may get downloaded via direct cable (USB). The installation procedure for downloaded applications onto phone/removable memory follows the below steps and rules:

- Initially, the user has a DOWNLOAD option that allows to choice between installing the application on removable memory or phone. The phone option should be the default option.
- There will be a separate directory within removable memory for Java ME applications. All Java ME Applications stored on removable memory and

information associated with them shall reside in this directory.
- An installed application on one device cannot be run on another device by swapping memory card. Separate installation is required for each device.
- Memory full condition handling while installing will be same for phone and removable memory.
- It is to provide push registry support for applications residing on removable memory. If this feature is not implemented, application that declare push registry usage in JAD file will not be allowed to be installed on removable memory.

15.3 Listing and Launching Java ME Applications from removable memory

By default, the JAM will list all installed applications on phone. The following rules will guide the user through of the available options:

- There will be a "Switch Storage Device" option under Games & Apps (MyJavaApps) menu allowing user to switch between storage devices (phone/removable memory) while listing applications. If user selects removable memory option, all installed applications on removable memory must get listed.
- Delete All Apps option under Java Settings menu must be effective for current storage device specified as above only. If phone is current storage device, only Java ME applications installed in phone must be deleted. The delete confirmation notice shall be modified to provide the current storage device information to the user. Delete all operation should only uninstall the application installed on removable memory. Original JAD and JAR files should not be deleted.
- Last menu item in the applications listing from removable memory shall be named [Install New]. If user selects this item, all application files (including already installed applications) from the Java ME directory in removable memory shall be listed. Both JAD and JAR files shall be listed. JAD and JAR file names will be preceded with a distinct icon to distinguish each type of file.
- User can select either JAD or JAR file if both are available. If user selects JAR file, implementation will search for JAD file with the same name in the same directory. If a JAD file is found and it refers to the selected JAR file, it shall be used. Otherwise JAR only installation shall be followed.
- Corresponding JAD and JAR files will be available in removable memory for installable applications requiring JAD file. JAD file should refer to the

local JAR file only. Downloading of JAR files over the network for new application installation shall not be supported for this release.

- All externally loaded application JAD and JAR files must be kept in the Java ME directory in removable memory for AMS to list these as new applications for installation.
- Context sensitive menu of new applications should have "Delete" option to delete the application files. This operation should delete both JAD and JAR files (if present) permanently from removable memory.
- JAM will refresh application listing from removable device each time listing is required after a power cycle. This is to ensure that applications removed/added externally can be reflected upon each invocation.
- AMS will do an extra preverification step before launching applications from removable memory. This is to ensure that applications are not corrupted (externally or otherwise). If application is corrupted, a prompt shall be displayed after the user selects application to be launched.
- Delete, Details, Permissions options should be available under Games & Apps Menu for Java ME Applications installed on removable memory. Delete operation should only uninstall the application installed on removable memory.
- AMS should support same application being installed on phone and removable memory. However duplicate applications will not be permitted on same storage device.
- If push registration is not supported for applications stored on removable memory, an IOException shall be returned if a Java ME application residing on removable memory tries to use javax.microedition.io.PushRegistry.registerConnection() method. The handling of this exception is left to the application.

16
JSR-185 - JTWI

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

16.1 Overview

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

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

Any Motorola JTWI device will implement the following and pass the corresponding TCK:

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

16.2 CLDC related content for JTWI

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

The following CLDC requirements will be supported:

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

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

- When creating a TimeZone, the specified custom time zone ID is normalized in the following syntax:
    - NormalizedCustomID:
        - GMT Sign TwoDigitHours: Minutes
        - Sign: one of:
            - + -
        - TwoDigitHours:
            - Digit Digit
        - Minutes:
            - Digit Digit
        - Digit: one of:
            - 0 1 2 3 4 5 6 7 8 9

16.3 MIDP 2.0 specific information for JTWI

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

- Record Store Minimum will permit a MIDlet suite to create at least 5 independent RecordStores. This requirement does not intend to mandate that memory be reserved for these Record Stores, but it will be possible to create the RecordStores if the required memory is available.
- HTTP Support for Media Content will provide support for HTTP 1.1 for all supported media types. HTTP 1.1 conformance will match the MIDP 2.0 specification. See package.javax.microedition.io for specific requirements.
- JPEG for Image Objects - ISO/IEC JPEG together wil JFIF will be supported. The support for ISO/IEC JPEG only applies to baseline DCT, non-differential, Huffman coding, as defined in JSR-185 JTWI specification, symbol 'SOF0'. This support extends to the class javax.microedition.lcdui.Image, including the methods outlined above. This mandate is voided in the event that the JPEG image format

becomes encumbered with licensing requirements.

- Timer Resolution will permit an application to specify the values for the firstTime, delay, and period parameters of java.util.timer.schedule () methods with a distinguishable resolution of no more than 40 ms. Various factors (such as garbage collection) affect the ability to achieve this requirement. At least 80% of test attempts will meet the schedule resolution requirement to achieve acceptable conformance.
- Minimum Number of Timers will allow a MIDlet to create a minimum of 5 simultaneously running Timers. This requirement is independent of the minimum specified by the Minimum Application Thread Count.
- Bitmap Minimums will support the loading of PNG images with pixel color depths of 1, 2, 4, 8, 16, 24, and 32 bits per pixel per the PNG format specification. For each of these color depths, as well as for JFIF image formats, a compliant implementation will support images up to 76800 total pixels.
- TextField and TextBox and Phonebook Coupling - when the center select key is pressed while in a TextBox or TextField and the constraint of the TextBox or TextField is TextField.PHONENUMBER, the names in the Phonebook will be displayed in the "Insert Phonenumber?" screen.
- Supported characters in TextField and TextBox - TextBox and TextField with input constraint TextField.ANY will support inputting all the characters listed in JSR-185.
- Supported characters in EMAILADDR and URL Fields - Class javax.microedition.lcdui.TextBox and javax.microedition.lcdui.TextField with either of the constraints TextField.EMAILADDR or TextField.URL will allow the same characters to be input as are allowed for input constraint TextField.ANY
- Push Registry Alarm Events will implement alarm-based push registry entries.
- Identification of JTWI via system property - to identify a compliant device and the implemented version of this specification, the value of the system property microedition.jtwi.version will be 1.0

16.4 Wireless Messaging API 1.1 (JSR-120) specific content for JTWI

WMA defines an API used to send and receive short messages. The API provides access to network-specific short message services such as GSM SMS or CDMA short messaging. JTWI will support the following as it is outlined in the JSR-120 chapter of this developer guide:

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

16.5 Mobile Media API 1.1 (JSR-135) specific content for JTWI

The following will be supported for JTWI compliance:

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

16.6 MIDP 2.0 Security specific content for JTWI

- The Motorola C975 follows the security policy outlined in the Security chapter of this developer guide.

17 JSR-184 - Mobile 3D Graphics API

17.1 Overview

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

17.2 Mobile 3D API

The Motorola C975 contains full implementation of JSR-184 Mobile 3D API (*http://jcp.org/en/jsr/detail?id=184*). The Motorola C975 has also implemented the following:

- Call to `System.getProperty` with key - microedition.m3g.version will return 1.0, otherwise null will be returned.
- Floating point format for input and output is the standard IEEE float having an 8-bit exponent and a 24-bit mantissa normalized to 1.0, 2.0.
- Implementation will ensure the Object3D instances will be kept reference to reduce overhead and possible inconsistency.
- Thread safety.
- Necessary pixel format conversions for rendering output onto device.

- Support at least 10 animation tracks to be associated with an Object 3D instance (including animation controller) subject to dynamic memory availability.

17.3 Mobile 3D API File Format Support

The Motorola C975 supports both M3G and PNG file formats for loading 3D content. The C975 supports the standard .m3g and .png extensions for its file formats. Mime type and not extension will be used for identifying file type. In the case that the Mime type is not available, M3G files will be identified using the file identifier and PNG files using signature.

17.4 Mobile 3D Graphics - M3G API

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

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

17.4.1 Typical M3G Application

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

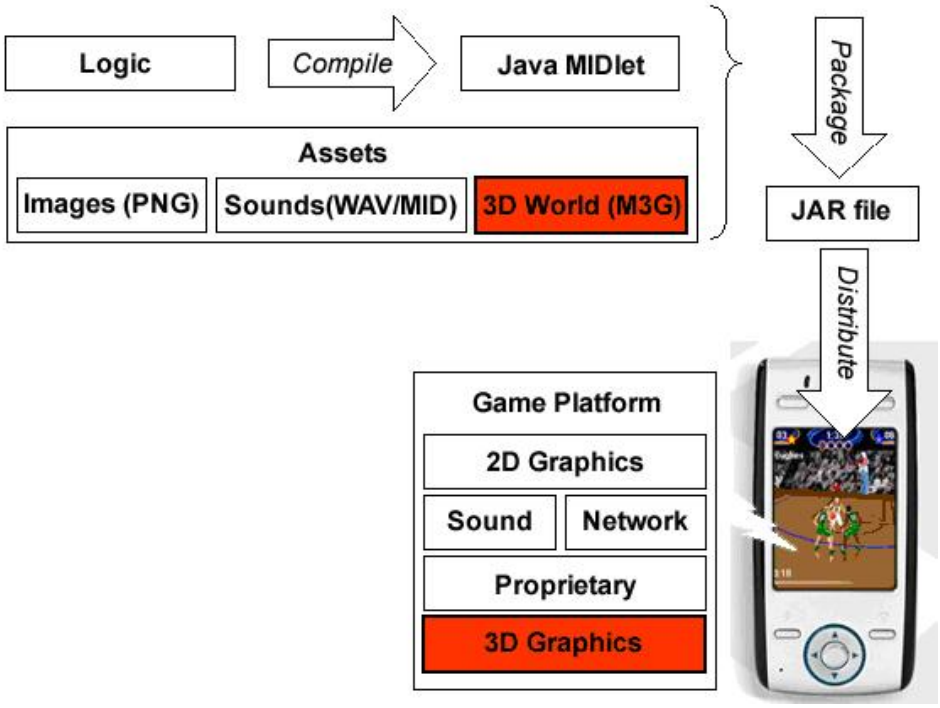

Figure 4 M3G Application Proccess

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

17.4.2 Simple MIDlets

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

Figure 5 M3G Application Methods

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

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

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

Figure 6 shows the structure of a more typical MIDlet.

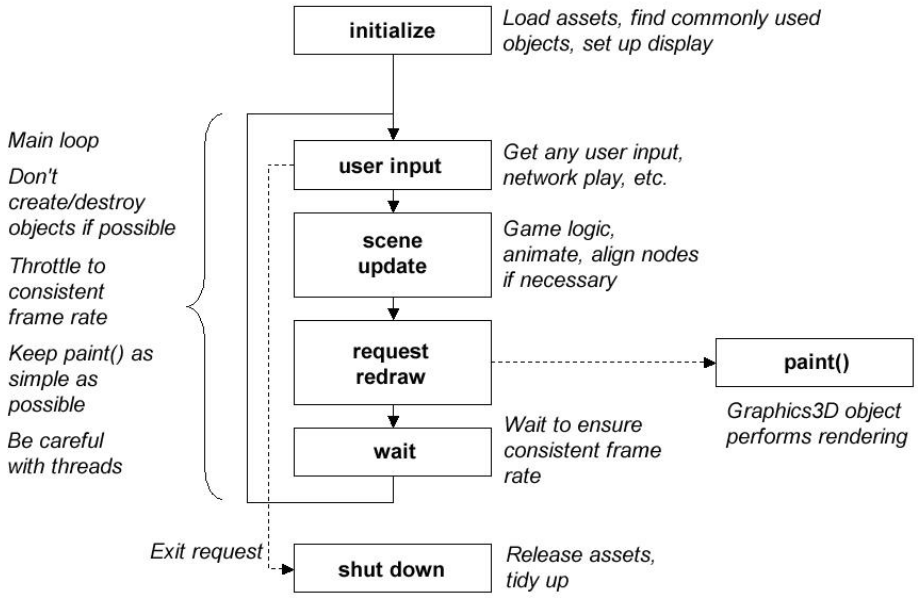

Figure 6 Typical MIDlet Structure

17.4.3 Initializing the world

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

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

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

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

The example assumes that there is only one root node in the scene, which will be the world object. If the M3G file has multiple root nodes the code must be changed to reflect this, but generally most M3G files have a single root node.

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

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

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

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

Code Sample 7 Initializing the world

17.4.4 Using the Graphics3D object

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

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

        public void paint(Graphics g)
        {
                myG3D.bindTarget(g);

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

        }
```

```
}
```

Code Sample 8 Using the Graphics3D object

The final block makes sure that the target is released and the Graphics3D can be re-used. The bindTarget call must be outside the try block, as it can throw exceptions that will cause releaseTarget to be called when a target has not been bound, and releaseTarget throwing an exception.

17.4.5 Interrogating and interacting with objects

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

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

Code Sample 9 Finding objects by ID.

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

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

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

		obj.getReferences(objArray);
```

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

        }

}
```

Code Sample 10 Using the Object3D.getReferences().

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

17.4.6 Animations

As well as controlling objects from code, scene designers can specify how objects should move under certain circumstances, and store this movement in 'canned' or block animation sequences that can be triggered from code. Many object properties are animatable, including position, scale, orientation, color and textures. Each of these properties can be attached to a sequence of keyframes using an AnimationTrack. The keyframe sequence can be looped, or just played once, and they can be interpolated in several ways (stepwise, linear, spline).

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

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

The animate() method returns a validity value that indicates how long the current value of a property is valid. Generally this is 0 which means that the object is still being animated and the property value is no longer valid, or infinity when the object

is in a static state and does not need to be updated. If nothing is happening in the scene, you do not have to continually redraw the screen, reducing the processor load and extending battery life. Similarly, simple scenes on powerful hardware may run very fast; by restricting the frame-rate to something reasonable, you can extend battery life and are more friendly to background processes.

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

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

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

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

Code Sample 11

17.4.7 Authoring M3G files

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

Appendix A
Key Mapping

Key Mapping

Table 26 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 z#)	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.

Table 26 Key Mapping

Appendix B Memory Management Calculation

The available memory on the C975 is the following:

- 4 MB shared memory for MIDlet storage
- 1.5 MB Heap size

Appendix C
FAQ

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

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

http://developer.motorola.com/

Appendix F
Spec Sheet

Spec Sheet

Listed below is the spec sheets for the C975. The spec sheet contains information regarding the following areas:

- Technical Specifications
- Key Features
- Java ME Information
- Motorola Developer Information
- Tools
- Other Related Information

Motorola C975
Developer Reference Sheet

Technical Specifications

Band/Frequency	UMTS 2100 MHz GSM 900/1800/1900 MHz GPRS (2U/4D, Class 10, B)
Region	North America
Technology	WAP 2.0, Java ME, SMS, EMS, MMS
Connectivity	USB, via CE Bus
Dimensions	53.2 x 114 x 24.2 mm
Weight	139 g
Display	1.9" 176 x 220 65k TFT Color
Operating System	Motorola
Chipset	

Key Features

- 3D stereo sound
- Point to Point Video
- Integrated Digital Video/Still Camera
- Large Color Display
- Integrated MP3 Player
- iTAP Predictive Text Entry
- Transflash expandable memory

Java ME Information

CLDC v1.1 and MIDP v2.0 compliant	
Heap Size	1.5 MB
Maximum record store size	64 KB
MIDlet storage avaliable	4 MB
Interface connections	HTTP 1.1, UDP, TCP
Maximum number of Sockets	4
Supported image formats	GIF,JPEG,PNG,BMP
Double buffering	Supported
Encoding schemes	ISO8859_1, ISO10646
Input methods	Multitap, iTAP
Additional API's	JSR-118 JSR-120 JSR-135 JSR-139 JSR-184 JSR-185
Audio	MIDI, MP3, AMR, WAV, MP4, iMelody

Related Information

Motorola Developer Information:
Developer Resources at
http://developer.motorola.com/
Tools:
Motorola Java™ ME SDK version v6.1 SE
Motorola Messaging Suite v1.1
Documentation:
Creating Media for the Motorola C975 Handset

References:
Java ME specifications:
http://java.sun.com/javame/
MIDP v2.0 specifications:
http://www.java.sun.com/products/midp
CLDC v1.0/v1.1 specifications:
http://www.java.sun.com/products/cldc
WAP forum: http://www.wap.org
EMS standards: http://www.3GPP.org
Purchase:
Visit the Motodev Shop at http://developer.motorola.com/
Accessories: http://www.motorola.com/consumer

Some features are network, subscription and SIM card or service provider dependent and may be not available in all areas. MOTOROLA and the Stylized M Logo are registered in the US Patent & Trademark Office. The Bluetooth® word mark and logos are owned by the Bluetooth SIG, Inc. and any use of such marks by Motorola is under license. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other product or service names are the property of their respective owners. © Motorola, Inc. 2006

Updated in 13-Jul-2006.

Appendix H
Quick Reference

CLDC: 9 16 16 18 65 87 87 96 96 100

HTTP: 32 32 83

JAD: 21 23 23 23 23 24 35 35 57 58 58 58 59 59 60 60 104

JSR-118: 21 24 35

JSR-120: 64 64 65 66 66 66 68 98

MIDP: 16 16 16 16 18 21 24 27 29 51 52 52 52 56 59 60 85 95 97 100 101 103

SMS: 23 27 27 34 41 64 64 65 66 66 66 66 67 67 67 67 71 98

WMA: 64 64 98

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

© Motorola, Inc. 2005-2006.