Java MIDlet Developers Guide for Cisco Unified IP Phones

Table of Contents

1 Overview -

Java MIDlet Developers Guide for Cisco Unified IP Phones provides the information you must understand and develop Java MIDlet applications for Cisco Unified IP Handheld Wireless Phones.

To develop Java MIDlet applications for Cisco Unified IP Handheld Wireless Phones, you must have the following resources:

- Cisco wireless IP Phones 7925G or 7926G
- Firmware load 1.4.1 and later, or a Developer version of 1.4.1 provided by Cisco
- Cisco Unified Communication Manager 7.0 or later to provision the Java applications

This document will continually be updated as we receive feedback, and ideas on how to improve it. Feel free to provide feedback to the phone team if you have suggestions on ways to improve it, or any tips that would make life easier for other developers.

2 Audience

Network engineers, system administrators, or telecom engineers should review this guide to learn about the Java MIDlet support on Cisco Unified IP Phones. General background knowledge of Java and the Mobile Information Device Profile (MIDP) is assumed and not covered in this document.

3 Related Documentation

For more information about Java MIDlet application support on Cisco Unified IP Phones, refer to the following links:

MIDP 2.0 Information and API Links:

- http://java.sun.com/products/midp/index.jsp
- http://jcp.org/aboutJava/communityprocess/final/jsr118/

Recommended Books

- http://java.sun.com/docs/books/midp/

- http://java.sun.com/docs/books/j2mewireless-2ndEd/

Obtaining Documentation and Submitting a Service Request

Please use the CDN forums for JMAPI as the location to ask questions, and provide feedback.

http://developer.cisco.com/web/jmapi/forums

Members of the phone team and Developer Support are subscribed to the forums and will be notified when questions are entered. We will strive to answer questions within 24 hours.

Members of the forum can receive notifications of updates by subscribing to the update service, which can be delivered either through email notifications or an RSS stream.

4 An Overview of Java MIDlet Application Support on Cisco Unified IP Phones -

This document describes the Java MIDlet application support on Cisco Unified IP Phones. General background knowledge of Java and the Mobile Information Device Profile (MIDP) is assumed and not covered in this document.

Running Java MIDlet applications directly on the phone allows more sophisticated application capabilities than with existing XML services, such as animated graphics, custom user interface objects, advanced network connectivity, and persistent local storage. Because Java MIDlets are an industry standard, developers can use standard Java development tools for building applications.

Cisco Unified Communications Manager IP Phone Service provisioning and subscription interfaces (CUCM Admin and CUCMUser) have been enhanced to allow for explicit provisioning of IP Phone Services via the phone's configuration file. This new explicit provisioning system allows Java MIDlet applications to be installed and managed on the phone. When a new service is added in the configuration, the phone downloads and installs it. When a service is removed, the phone uninstalls it.

Cisco Unified Communications Manager 7.0(1) is the earliest version that introduces this enhanced service provisioning and no earlier version is able to provide this service provisioning. Each target platform (Cisco Unified IP Phone) will also have a minimum version of firmware required in order to complement the service provisioning.

5 Supported Cisco Unified IP Phone platforms

Each of the following platforms provide support for Java MIDlet applications. But not all

platforms are available for general development. At the time of writing, only the 7925G and 7926G are available without restrictions.

Model Number		Display Type	Navigation	Touchscreen Support	Barcode Scanner
7925G		Color	4-way plus Select	No	No
7926G		Color	4-way plus Select	No	Yes

6 Supported Features

The Java MIDlet implementation supports the following Java specifications:

Mobile Information Device Profile (MIDP) version 2.0

Connected Limited Device Configuration (CLDC) version 1.1

No other optional Java JSRs are supported in this release—only the core MIDP and CLDC APIs.

7 Optional Java MIDlet Features

The Java MIDP API specification declares support for some features and components as optional, therefore each implementation must document whether or not those components are supported.

Table 3-1 documents the level of support provided for optional MIDP components on Cisco Unified IP Phones. Unless otherwise specified in the Notes column, the term “Supported” applies to all phone models and firmware

Table 3-1 Supported Java MIDlet Features

Component	Feature	Support	Notes

Micro IO Networking (javax.microedition. io)	HTTP client connections	Yes	—
	TCP client and server connections	Yes	—
	UDP client and server connections	Yes	Multicast server connections are also supported by providing a Cisco-specific attribute of “mode=server” on the URI. For example: datagram://239.1.2.3:12345;mo de=server
	TLS client connections	Yes	—
	HTTPS client connections	Yes	—
	PushRegistry support	Yes	Both Connection (sockets and datagrams) and Alarms entries are supported
Media (javax.microedition. media)	Sampled Audio Player	Yes	Max audio data size of 250 Kbytes Down-sampled to 8kS/s, mono before playback Only supported content types: • RAW (no file header information) uLaw - 8kS/s, 8-bit, • single channel (mono) WAV files: • uLaw: 8kS/s, 8 bit, single channel (mono) • PCM: 8-44 kS/s, 8-16 bit,

			multi-channel

Table 3-1 Supported Java MIDlet Features (continued)

Component	Feature	Support	Notes
LCDUI (javamax.microedition.lcdui)	Pointer events	Yes	Press, drag, and release events are supported on all phones that have a touchscreen or other pointer device
	Pointer motion events	Yes	
	Key repeat events	Yes	—
	Vertical navigation	Yes	—
	Horizontal navigation	Yes	For all devices which have a 4-way or 5-way navigation cluster
	Double-buffered rendering	Yes	Supported on all low-level render targets (Canvas, GameCanvas, and CustomItem)
	Alpha transparency in images	Yes	Both full transparency and semi-transparency

Note Only 8-bit PCM is supported in Cisco Unified Wireless IP Phone 7925G and 7926 phones.

7.1 Application and Window State Management

The MIDP API provides two methods for managing the application state and window state (focus)—the MIDlet object and the object.

Table 3-2 describes the different initial application and window states, the event that is called

by each MIDlet object and display object, other external events, and the resulting application and window states of the MIDlet. Events that do not effect applications states and window states are not listed.

Table 3-2 Java MIDlet Features

Initial Application State	Initial Window State	Event	Resulting Application State	Resulting Window State
Active	In Focus	MIDlet.notifyPaused()	Paused	Minimized
		Display.setCurrent(null)	Paused	Out of Focus
		Another application gains focus	Paused	Out of Focus
Paused	Out of Focus	MIDlet.resumeRequest()	Active	In Focus
		Display.setCurrent(non-null)	Active	In Focus
		Another application loses focus	Active	In Focus
Paused	Minimized	MIDlet.resumeRequest()	Active	In Focus
		Display.setCurrent(non-null)	Active	In Focus
		Another application loses focus	Paused	Minimized

Table 3-3 lists the different MIDlet states for Cisco Unified Wireless IP Phone 7925 G and 7926 phones.

Table 3-3 Java MIDlet States

States	Event	Platform Behavior	MIDlet Application Requirements
Active	Minimize transition	Switch to GUI screen	-
	Pause Transition	Switch to GUI screen and pause the MIDlet	Stop resources used by the MIDlet

	Destroy Transition	Switch to GUI screen and destroy the MIDlet	Stop resources used by the MIDlet
	Answer Call	Pause	-
Minimized	Active transition	Switch to Java screen	-
	Pause Transition	Pause MIDlet and update MIDlet state on service options	Stop resources used by the MIDlet
	Destroy Transition	Destroy MIDlet and update MIDlet state on service options	Stop resources used by the MIDlet
	Answer Call	-	-

Table 3-3 Java MIDlet States (continued)

States	**Event**	**Platform Behavior**	**MIDlet Application Requirements**
Minimized with Resources	Active transition	Switch to Java screen	-
	Pause Transition	Pause MIDlet and update MIDlet state on service options	Stop resources used by the MIDlet
	Destroy Transition	Destroy MIDlet and update MIDlet state on service options	Stop resources used by the MIDlet
	Answer Call	Pause MIDlet	-
Paused	Active transition	Switch to Java screen and call startApp	Resume resources for the MIDlet
	Destroy Transition	Destroy MIDlet and update	Stop resources used by the

		MIDlet state on service options	MIDlet
	Answer Call	-	-
Destroyed	Active transition	Switch to Java screen and call startApp	Start the required MIDlet resources

Note The Display.setCurrent(null) requests the MIDlet window to lose focus only if there is another application indow open that can be displayed in the foreground to be in focus. When an application is minimized, it does not become visible again until it is explicitly brought back into focus, even if all other applications are closed.

7.2 Position of Softkeys

Cisco Unified Wireless IP Phone 7925G and 7926 phones do not support positioning of the softkeys .

7.3 MIDP Media

Cisco Unified Wireless IP Phone 7925G and 7926G phones do not provide support for Virtual Player.

7.4 System Properties

System Properties are environment properties which are common for all applications running on the phone and are retrieved by calling java.lang.System.getProperty(String propName). For example:

String myHostname = System.getProperty(“microedition.hostname”);

Table 3-5 lists the Java System Properties that are supported, along with sample return values.

Table 3-5 Java System Properties

System Property	Example Value	Notes
microedition.configurat ion	CLDC-1.1	—
microedition.profiles	MIDP-2.0	May contain multiple comma-separated values.
microedition.hostname	SEP000012345678	Multicast server connections are also supported by providing a Cisco-specific attribute of "mode=server" on the URI. Example: datagram://239.1.2.3:12345;mode=server
microedition.locale	en-US	—
microedition.encoding	iso-8859-1	This is the default single-byte encoding. Note that all MIDlet-enabled loads also support UTF-8.

Table 3-5 Java System Properties (continued)

System Property	Example Value	Notes
microedition.platform	Cisco 7975G/8.4.1 **Note** For Cisco Unified Wireless IP Phone 7925G and 7926 phones, the microedition platform returns "j2me".	(Model name) / (Supported API version). **Note** "j2me" for Cisco Unified Wireless IP Phone 7925G and 7926 phones.
com.cisco.device.settings.config.userid	userid	Cisco-specific property which returns the userid who is currently logged into the phone, or else the statically-associated owner userid in CUCMAdmin pages.
com.cisco.device.sett	000012345678	Cisco-specific property which returns

in gs. network.macaddress (javax.microedition.l cd ui)		the MAC address of the phone.
microedition.cisco.ip ph one.platform	Cisco CP-7925G/1.1.JAVA	(Model name)/(Supported API version).

7.5 Application Properties

The Java MIDP API requires that the MIDlet application be able to query for any application-specific properties which are contained in either the JAD descriptor file or the JAD manifest file via the javax.microedition.midlet.MIDlet.getAppProperty(String propName) method. For example:

String myVersion = myMidlet.getAppProperty("MIDlet-Version");

Cisco Unified IP Phones also support a second, Cisco-specific, mechanism for providing Application Properties to a MIDlet (in addition to the standard JAD/JAR mechanism). When query-string parameters that are added to the end of the Phone Service URL (which points to the JAD file) they are also added as Application Properties when the MIDlet is launched. This feature was added for the following reasons:

- To maintain consistency with how XML services are configured and provisioned
- To allow the user to provide parameter values when subscribing to a MIDlet service
- To allow an admin to provide "bootstrap" configuration parameters to a MIDlet directly from the CUCMAdmin pages, rather than having to manually modify the JAD file.
- Changes to JAD properties are not automatically propagated to the phones and is just an HTTP download without any change notification mechanism. However changes in service parameters, are propagated to the phones and they can update their settings without further intervention from the administrator.

  For example, the following Phone Service URL allows "server1" and "server2" values to be queried by the MIDlet at runtime, just as if they had appeared in the JAD file:

  http://myserver/midlets/MyMIDlet.jad?server1=foo&server2=bar

  The following example would allow the Weather MIDlet to query for a ZIP code parameter named "zip" which was entered when the user subscribed to the service:

  http://myserver/midlets/WeatherMIDlet.jad?zip=62025

7.6 Multi-User RMS Storage

Cisco Unified IP Phones support the Extension Mobility feature, which allows multiple users to log in to the same physical phone and load the user's settings (and MIDlet applications) onto the phone. Because this feature makes the phone a multi-user device, the proper security must be in place to ensure that user-specific data is not accessible by other users.

Note RMS record stores can store up to 2 MB for Cisco Unified Wireless IP Phone 7925G and 7926 phones.

The mechanism in place to protect user-specific data basically treats the RMS stores as cache files, so that when the phone's current user changes, all previous user data (installed MIDlets, and RMS data stores created by those MIDlets) are deleted. The implication to MIDlet developers is that RMS record stores cannot be relied upon as a method of permanently persisting data, especially in deployments where Extension Mobility is likely to be used.

MIDlet suites are uninstalled and their RMS record stores, if present, are deleted if any of the following events occur:

- If Extension Mobility is used to change the current active user of the phone (occurs in login and logout)
- If the phone registers to a different Unified CM cluster which does not support MIDlets or has a different services configuration for the device
- If nobody is logged into the phone via Extension Mobility, but the static "owner" of the phone is changed in Unified CM Admin configuration (this changes the current active user of the phone)
- If the configuration is cleared from the phone using the Settings menu or by any other means (such as a factory reset).

7.7 Phone Web Pages

Cisco Unified IP Phones have an embedded web server which is capable of serving device state and configuration information via HTTP. While these web pages are not specific to the MIDlet capabilities, they can be useful since the MIDlet is running locally on the device and can access them by simply connecting to "localhost".

The following link to the API specification lists the available web pages:

http://www.cisco.com/en/US/docs/voice_ip_comm/cuipph/all_models/xsi/8_0_1/httpsettings.html#wp 1034495

7.8 Platform Requests –

As per MIDP specification, a MIDlet is allowed to invoke platform-specific features via the MIDlet.platformRequest(String uri) method, and Cisco Unified IP Phones support the following URIs:

- RTP Streaming URIs: Initiating and terminating RTP audio streams to/from the phone
- Enhanced Dial URI: Initiating new calls and controlling the user interface for a call
- SendDigits URI: Injecting DTMF digits into active calls

These URIs are documented in the *Phone Firmware 8.4(x) - IP Phone Services Enhancements* documentation bundle that can be downloaded from the Cisco Developer Support site and also at Cisco IP Phone Services Development Notes 7.0(1) or later.

http://developer.cisco.com/web/ipps

Additionally, for Cisco Unified Wireless IP Phone 7925 and 7926 phones, the following statements are true when using the platformRequest() API.

- When a call is active, neither Unicast nor Multicast RTP Transmit/Receive is initiated and vice versa.
- When Unicast RTP Transmit is active, a Multicast Transmit cannot be initiated and vice versa.
- When a Unicast or Multicast stream is active, another stream (Unicast or Multicast) cannot be initiated.
    - When a Unicast RTP Receive is active, a Multicast Receive cannot be initiated and vice versa.

In all of the above cases a ConnectionNotFoundException is thrown upon violation.

Applications implementing the Push-To-Talk (PTT) feature must be aware of the latency in RTP channel setup. It is recommended that the application intending to transmit using platformRequest(), should provide visual/audio indication to prevent any loss of audio.

7.9 Using Application IDs in MIDlets

When invoking any URIs which require an Application ID (such as the Dial and SendDigits URIs), the Application ID for a MIDlet must be specified in the form of *VendorName/SuiteName/MIDletName*.

For example, a MIDlet suite with the following JAD file would invoke the Dial URI in the LcduiMIDlet by using:

myMIDlet.platformRequest(“Dial:5551212:1:CiscoSystems/SampleMIDlets/LcduiMIDlet”);

MIDlet-Name: SampleMIDlets
MIDlet-Vendor: CiscoSystems
MIDlet-Version: 1.0
MIDlet-1: LcduiMIDlet, /icons/cisco_icon.png, cip.samples.lcdui.LcduiMIDlet
MIDlet-2: AudioMIDlet, , cip.samples.AudioMIDlet
MIDlet-3: GameMIDlet, /icons/gun.png, cip.samples.GameMIDlet
MIDlet-Description: Sample MIDlets from SDK
MIDlet-Info-URL: http://www.cisco.com/go/developersupport
MIDlet-Icon: /icons/cisco_icon.png

MIDlet-Jar-Size: 225998
MIDlet-Jar-URL: SampleMIDlets.jar
MIDlet-Permissions: javax.microedition.io.Connector.http,
javax.microedition.io.Connector.https

7.10 Key Codes –

The low-level MIDP LCDUI components (CustomItem, Canvas, and GameCanvas) provide low-level rendering and input handling which enables the creation of custom UI components. As per MIDP specifications, any compliant device must support at least a basic 12-key telephony keypad consisting of 0 through 9, *, and #. In addition to these standard key code definitions, Cisco Unified IP Phones also support the keys shown in Table 3-7.

Table 3-7 Supported Keys

Key	**Key Code**
NavUp	-1
NavDown	-2
NavLeft	-3
NavRight	-4
NavSelect	-5
Soft1	-6
Soft2	-7
Soft3	-20

Soft4	-21
Soft5	-32

As per MIDP specifications, standard MIDP keys use positive integer values and device-specific key codes are negative. With the exception of Soft Key 5 (which is not supported by the WTK), Sun's Wireless Toolkit emulator device uses the exact same key code values as listed above for Cisco Unified IP Phones. These were intentionally chosen to simplify the creation of custom UI components using the WTK.

Since all of the low-level UI components are also capable of supporting Commands which would use the softkeys, it might seem redundant to also define device-specific key codes for them and directly deliver low-level keys events. While it is true that Commands allow the MIDlet to control the softkeys, there are two limitations to this high-level interface which are overcome with the low-level key events:

- Command listeners only receive a generic "invoke" event, but key event listeners receive separate press and release events. This allows them to be used for "momentary" type functions such as push-to-talk, or key-repeat functions.
- Commands can only use text as the label and the label is required. Low-level key events combined with a full-screen mode (not yet supported in this releases) allows custom rendering of softkey labels, such as icons and alpha-blended overlays and backgrounds.

Cisco Unified Wireless IP Phone 7925G and 7926 phones have the PTT button. MIDP Applications, in their keyPressed(), keyReleased(), keyRepeated() methods, use com.cisco.midp.lcdui.KeyCodeMap.KEY_PTT.

These phones have Send(Green) and Hang Up(Red) keys. Send key behaves as a backspace key for text field, which is used to delete charaters during character input mode.
Hang Up key provides a way to minimize the midlet. When the application is minimized, it gives the display to the call plane. However, it keeps running in the background.

7.11 Game Actions

Since MIDP is most prominent on consumer electronics devices (i.e., cell phones), the low-level MIDP LCDUI components provide robust support for game development. One of those capabilities includes the concept of "game actions". Game actions allow common gaming controls to be defined to whatever keys are most appropriate for the specific device. Each device documents their key mapping for game actions and the MIDlet queries the platform dynamically to map a key event to a game action. Table 3-8 defines the key-to-game-action mapping for Cisco Unified IP Phones:

Table 3-8 Key-to Game-Action Mapping for Cisco Unified IP Phones

Key	Game Action
NavUp	Canvas.UP
NavDown	Canvas.DOWN
NavLeft	Canvas.LEFT
NavRight	Canvas.RIGHT
NavSelect	Canvas.FIRE
Canvas.KEY_NUM_0	Canvas.NO_ACTION
Canvas.KEY_NUM_1	Canvas.GAME_A
Canvas.KEY_NUM_2	Canvas.UP
Canvas.KEY_NUM_3	Canvas.GAME_B
Canvas.KEY_NUM_4	Canvas.LEFT
Canvas.KEY_NUM_5	Canvas.FIRE
Canvas.KEY_NUM_6	Canvas.RIGHT
Canvas.KEY_NUM_7	Canvas.GAME_C
Canvas.KEY_NUM_8	Canvas.DOWN
Canvas.KEY_NUM_9	Canvas.GAME_D
Canvas.KEY_STAR	Canvas.NO_ACTION
Canvas.KEY_POUND	Canvas.NO_ACTION

1 A	2 UP	3 B
4 LEFT	5 FIRE	6 RIGHT
7 C	8 DOWN	9 D
*	0	#

7.12 Font Support

The User Interface on Cisco Unified IP Phones is strictly controlled to maintain the Cisco look and feel. To minimize the impact on supporting existing devices, font support in the Java MIDP API is much more limited than in traditional desktop GUI environments. For example, Java MIDP API does not allow applications to load custom fonts or dictate specific (numeric) point sizes. Instead, the Java MIDP API is structured so that MIDlet applications typically reuse and share the fonts that are already used by the native user interface of the device in order to save memory usage.

The Java MIDP Font API provides access to font objects in two different ways:

- High-level getter methods—These return fonts already used by the phone.
- Low-level getter methods—These return fonts that match a specific set of attributes.

Table 3-9 describes which fonts are returned from each of these Java MIDP Font API methods, and how those fonts relate to the standard phone fonts used by the embedded user interface on the phone.

Note Font point sizes and other attributes of the standard fonts used by the phone vary by phone model and user locale. Because of this, there are no *specific* attribute details that can be defined—only high-level rules of how the Java MIDP Fonts relate to those standard phone fonts.

Java MIDP Font API Method	Font Description

getDefaultFont() getFont(FONT_STATIC_TEXT)	Returns a SIZE_MEDIUM font that is normally used by the phone to display titles and static text.
getFont(FONT_INPUT_TEXT)	Returns a SIZE_LARGE font that is normally used by the phone to display items in a menu or an editable input field.
getFont(int face, int style, int size)	Face—Only one font face is supported, so this parameter will be ignored. Style—All MIDP style flags are supported **Note** The phone supports additional style flags beyond what MIDP provides. Therefore, the standard FONT_STATIC_TEXT and FONT_INPUT_TEXT fonts returned in the high-level methods cannot be exactly reproduced by the low-level method, which only accepts the standard MIDP attributes. Size—MEDIUM and LARGE returns a font that is the same point size as the standard STATIC_TEXT and INPUT_TEXT fonts, respectively. SMALL uses a mode-specific point size that is less than or equal to the MEDIUM point size.

Cisco Unified Wireless IP Phone 7925G and 7926G phones support fonts only in English and the default font is dislayed with STYLE BOLD. This is to ensure that the characters stand out on the screen of the Cisco Unified Wireless IP Phone 7925G and 7926G. The characters within the Call UI of the Cisco Unified Wireless IP Phone 7925G and 7926G are displayed in bold, and if the characters in a Java MIDlet are not displayed in bold, they will appear thin and insipid.

There is not support for other styles. If the application specifies any other style, it is ignored and shown only as STYLE_BOLD.
All the font size, SMALL, MEDIUM and LARGE are supported.

8 Software Development Tools

8.1 Java IDEs for MIDP

Since Java MIDP and CLDC are in industry-standard APIs, there are many development tools

available in the market. Any standards-compliant tool can be used to for developing MIDlets on the Cisco IP Phones.

One of the most popular Java Integrated Development Environments (IDEs) is NetBeans. This is an open-source Java-based IDE owned by Sun Microsystems. It runs on Windows, Linux, Mac OS X, and Solaris. NetBeans can be downloaded for free from the following URL:

http://download.netbeans.org/netbeans/

Note
Make sure you download a bundle which includes the "Java ME" feature to support Java MIDlets.

The NetBeans 6.8 Java bundle includes Sun Microsystem's JavaME SDK which is a MIDP-emulator and debugging environment. You can also use the latest JavaME SDK independent of the NetBeans bundle by downloading it from the following URL:

http://java.sun.com/javame/sdk/

When developing Java MIDlets (or other Java applications), you should obfuscate your code to make it more difficult for external parties to reverse-engineer your Java class files (contained in the JAR) back into readable source code. The obfuscator renames class and method names with short, non-descriptive names (typically just single letters) which not makes it difficult to read, but also decreases the size of the class files. (Just the opinion of some developers, but not to be construed as the official Cisco position)

Most Java IDE's and CLI build tools have integrated obfuscation tools, but several free open-source obfuscators are also available.

8.2 Emulator Skins for Cisco Unified IP Phones

Sun Microsystems's JavaME SDK has a "skinning" capability which allows device vendors to create skins so the emulator looks and behaves more like their actual physical device (display size, color/grayscale, bit-depth, touch-screen, soft keys, hard keys, navigation, etc.). Using skins, two models of the Cisco Unified IP Phones can be emulated reasonably well. The models of skin available are the 7975 and 7925/26. There are still some minor differences, such as Select softkey support, Ticker behavior and location. The Cisco MIDlets SDK provides skins for these IP Phones that support MIDP.

NOTE: Most Cisco-specific APIs, platform requests, and call controls are not supported by the emulators. However, the ScannerAPI is supported for the 7926 Phone via an included project.

Please see the "Cisco 7926 Emulator Skin Guide", included with the Cisco MIDlets SDK.

8.3 Cisco Unified IP Phone Simulator

The Cisco Unified IP Phone Simulator is a tool for enabling developers to write phone model specific applications. The Cisco Unified IP Phone Simulator runs the same core Java code as the hardware based phones. This device requires a Cisco Unified Communication Manager environment to operate. A production version of the Cisco Unified IP Phone Simulator exists in the form of the Cisco Unified IP Communicator product.

IP Phone Simulator vs. IP Communicator

The Cisco Unified IP Phone Simulator has several unique features that are not present in the Cisco Unified IP Communicator. First, the Cisco Unified IP Phone Simulator has no visual representation other than the screen portion of the phone. Second, the Cisco Unified IP Phone Simulator is available for Cisco Unified IP Phone 7911/06, 7941/61, 7970/71, 7962, 7965 and 7975. Not all of these IP Phones support Midlets, but can be used for testing traditional Cisco Unified IP Phone applications. The Cisco Unified IP 7975 Simulator registers and consumes licenses as a Cisco Unified IP Phone 7975. The Cisco Unified Communications Manager cannot distinguish between an IP Phone Simulator and the hardware version of that phone.

Installing the Cisco Unified IP Phone Simulator

The hardware requirements for the Cisco Unified IP Phone Simulator are the same as the requirements for Cisco Unified IP Communicator.

For information about the hardware requirements for IP Communicator, see

http://www.cisco.com/cisco/web/support/index.html

The Cisco Unified IP Phone Simulator will be initially available as a ZIP file. The only prerequisite is a working copy of Cisco Unified IP Communicator. Configuring the Cisco Unified IP Communicator creates the correct Windows registry keys, installs several critical components and also tunes the audio settings. ****STILL CORRECT****ALSO, MAKE THIS FORMATTED STEPS LIKE BELOW****

Using the Cisco Unified IP Phone Simulator

To use the Cisco Unified IP Phone Simulator:

Step 1 Open a command prompt or create a shortcut that points to the SoftPhone.bat

file.

Step 2 From a command prompt enter “SoftPhone” followed by one of these model numbers: 7911, 7941, 7970, 7941, 7942, 7945 and 7975.

For example: C:\SoftPhone>SoftPhone 7975

A second command window opens and acts as the console for the running Cisco Unified IP Phone Simulator. The user interface appears, and the phone attempts to register with the TFTP server configured during the IP Communicator setup process. Like a hardware phone, the Cisco Unified IP Phone Simulator can be configured for you to use DHCP Option 66 or 150 to select the TFTP server, or it can be configured using the alternate TFTP setting.

Each model has a unique MAC address that enables each model to be uniquely configured in Cisco Unified Communications Manager. If auto registration is enabled, the phone auto registers as the device it represents. During registration and normal operations, the Cisco Unified IP Phone Simulator may “reboot”.

Step 3 Close the UI and Console windows when the Cisco Unified IP Phone Simulator reboots.

Step 4 Restart the SoftPhone using the SoftPhone.bat tool.

Caution Currently only one instance of a Cisco Unified IP Phone Simulator may be running on a machine. If a second instance is started, it might cause a lock up that will require a hard restart (power off) which can result in loss of data.

In some cases, the Cisco Unified IP Phone Simulator may enter a state in which it behaves as if the Ctrl key is being pressed when it is in fact not being pressed. The Cisco Unified IP Phone Simulator will remain functional, but the Ctrl key does not need to be pressed. Other key combinations might not work when the Cisco Unified IP Phone Simulator is in this state. In this case, you must stop and restart the Cisco Unified IP Phone Simulator.

Table 4-1 lists the PC keyboard shortcuts for the Cisco Unified IP Phone keys.

Table 4-1 Keyboard Shortcuts

Phone Keys	**PC Keys**

Phone Line Keys	Ctrl Plus Line Number - Ctrl+1
Keypad	Numbers 0-9, * and #
Directory	Ctrl+D
Settings	Ctrl+S
Services	Ctrl+V
Messages	Ctrl+M

Table 4-1 Keyboard Shortcuts

Phone Keys	**PC Keys**
Information	Ctrl+I
Headset	Ctrl+H
Speaker	Ctrl+P

9 Software Development Kit

The Cisco IP Phone Services MIDlet Software Development Kit (SDK) contains everything that you will need to create MIDlet applications; including necessary documentation and sample applications. To obtain the SDK, contact Cisco Developer Services at:

http://developer.cisco.com/web/jmapi

Specifically, the SDK includes three sample applications. These applications are provided as training material for developers on the Cisco Unified IP Phone environment. The applications included with the SDK are Scanner Sample, Image Demo, and Device Specifics. Each application showcases specific features or capabilities of the Cisco Unified Wireless IP Phones and provides examples for implementing functionality in your own application. The applications are provided as NetBeans projects. For more details on their usage, see the guides included with this SDK.

9.1 Scanner Sample MIDlet

The Scanner Sample MIDlet provides you with useful examples and directions for incorporating

the barcode reader into your applications. This project shows you how to access the required APIs for starting, stopping, and receiving results from the barcode reader.

9.2 Image Demo MIDlet

The Image Demo MIDlet provides you with useful examples and directions for incorporating images into your applications. The project includes a basic image rendering library which can manage the storage and retrieval of images. This enables easy access to images without creating global variables and ensures efficient use of memory. The library also enables caching of images as they are downloaded from a web resource.

9.3 Device Specifics MIDlet

The Device Specifics MIDlet provides you with useful examples and directions for incorporating the device specific information and call control platform requests into your applications. This project shows you how to access information like platform and version information. In addition, this program shows you how to initiate and terminate a voice call from within your midlet.

10 Java MIDlets

A Java MIDlet Suite is the unit of installation for Java MIDP applications. A single MIDlet Suite consists of a Java Application Descriptor (JAD) file and a Java Archive (JAR) file. The JAD file contains a description of the MIDlet applications (classes) which are contained in the JAR file. Each MIDlet Suite contains one or more Java MIDlet applications. A MIDlet application is a subclass of javax.microedition.midlet.MIDlet that is present in the JAR file and is declared to be an application in the JAD file. So the unit of execution is a single MIDlet class from the MIDlet Suite.

The 7925G and 7926G Cisco Unified Wireless IP phones do not require a Java MIDlet to be signed by Cisco in order for the Java MIDlet to run. The 7925G and 7926G Cisco Unified Wireless IP phones do not support signing, and there are no restrictions on the Java MIDlets that can be executed by the phones, other than any platform and memory restrictions that may exist.

11 Provisioning Java MIDlet Applications -

Java MIDlets are provisioned to Cisco Unified IP Phones using the same Cisco Unified IP Phone Services mechanisms as XML services. They are two different types of Phone Services: XML services and Java MIDlet services.

Enhancements to the Phone Services provisioning mechanism were required in order to support Java MIDlets, and were introduced with Cisco Unified Communications Manager 7.0(1)

To take Java MIDlet application (in the form of a JAD file and a JAR file) and get it running on an IP Phone, perform these steps.

Procedure

Step 1 From the Cisco Unified Communications Manager Admin main page, choose **Device > Device Settings > Phone Service**s.

Step 2 Click **Add a New Service**.

Enter the service definitions into the appropriate fields. The following is an example of a MIDlet Phone Service configuration, as seen in the XML configuration file, based on the fields defined in the service definition

MIDlet-Name: SampleMIDlets

MIDlet-Vendor: CiscoSystems

MIDlet-Version: 1.0

MIDlet-1: LcduiMIDlet, /icons/cisco_icon.png, cip.samples.lcdui.LcduiMIDlet

MIDlet-2: AudioMIDlet, , cip.samples.AudioMIDlet

MIDlet-3: GameMIDlet, /icons/gun.png, cip.samples.GameMIDlet

MIDlet-Description: Sample MIDlets from SDK

MIDlet-Info-URL: http://www.cisco.com/go/developersupport

MIDlet-Icon: /icons/cisco_icon.png

MIDlet-Jar-Size: 225998

MIDlet-Jar-URL: SampleMIDlets.jar

MIDlet-Permissions:
javax.microedition.io.Connector.http,
javax.microedition.io.Connector.https

11.1 IP Phone Services Configuration Fields

For security reasons, the following Phone Service fields must match the MIDlet-Name and MIDlet-Vendor attributes in the JAD file, otherwise the MIDlet will not be allowed to install and

run on the phone.

Service Name
The Service Name field must be exactly the same as the MIDlet-Name: field inside the MIDlet JAD file. If there is any white space or mixed case in the MIDlet-Name field, this must be maintained in the Service Name otherwise the MIDlet will fail to start.

Service Version

The version must either match the MIDlet-Version attribute in the JAD file, or it can be left blank to indicate that a specific version is not required. In this case, whatever version is returned in the JAD is installed. Leaving the Service Version field empty enables an "auto-versioning" feature because the phone attempts to download the JAD file every time the phone re-registers to the Cisco Unified Communications Manager as well as every time the MIDlet is launched. This ensures that the phone is always running the latest version of the MIDlet without the administrator having to manually change the Service Version value. If the Version field is not empty, then the phone only downloads the JAD file and attempt to upgrade/downgrade the MIDlet if Version is different from what is currently installed on the phone.

Service Type

Type is a new field which allows Phone Services (Java MIDlets or XML services) to be provisioned to either the Services button (the default Standard IP Phone Service value) or to the Messages or Directories buttons. In previous releases, Phone Services could only be provisioned to the Services button.

Service Category

Category must be set to "Java MIDlet" rather than the default "XML service" value. This tells the phone that it is a Java MIDlet service and therefore it should download and install the MIDlet from the given URL.

Service URL

For a Java MIDlet, the URL must point to the location where the JAD file can be downloaded. This can be any generic web server, but in most cases it is the back-end application server which the MIDlet application would normally communicate with. For example, the application server that the Java MIDlet client was distributed with).

Note These MIDlet JAD and JAR files are not necessarily served to the IP Phone by the Cisco Unified Communications Manager server. The Cisco Unified Communications Manager simply provides a URL to the phone from which it can be downloaded.

ASCII Service Name

The ASCII Service Name is the field that will be displayed in the list of services visible by the phone, and has no impact on the MIDlet when it is loaded or run. This applies to

the Service Description field too.

Enable

The checkbox enables/disables the Phone Service. When a service is disabled, it is removed from the list of provisioned services sent to the phone and it does not appear in any phone menus. This checkbox is useful for temporarily disabling a service (possibly due to technical issues or during an upgrade) without having to delete the Phone Service configuration and add it again later and re-subscribe the devices.

Enterprise Subscription

If the Enterprise Subscription checkbox is selected, then this Phone Service is automatically deployed to all devices capable of supporting it. For example, a Java MIDlet service is deployed to all phone models capable of supporting Java MIDlets. This new feature allows a common service to be automatically deployed to all phones, avoiding the manual subscription (either by the admin or the end user) or scripted subscription (via BAT other automation utilities). If the Enterprise Subscription checkbox is deselected (default), then devices must manually subscribe to the Phone Service before the phone service appears on the phone.

Once the Java MIDlet Phone Services are configured and provisioned as described above, the IP Phone receives those Phone Services settings via its configuration file upon startup. The phone then compares the provisioned MIDlet Phone Services against the list of currently installed MIDlets to determine if any MIDlet suites need to be installed, uninstalled, upgraded, or downgraded, and automatically takes the necessary actions to reconcile.

The process by which the phone downloads the MIDlet JAD and JAR files and installs them onto the phone is compliant with the MIDP Over-The-Air (OTA) Provisioning Specification 1.0

The following URL provides more information about the OTA process and also links to additional resources:

http://developers.sun.com/mobility/midp/articles/ota/

The 7925G and 7926G Cisco Unified IP Wireless phones do not support this mechanism. The details of the reload mechanism are detailed in the following section.

If any MIDlet suites fail to properly install, the phone automatically retries using an exponential back-off algorithm, so the phone does not require additional resets/restarts to overcome transient configuration, server, or network problems. The phone continually retries to install the application, beginning at an interval of approximately 2 minutes, and increasing to a maximum interval between retries of approximately 128 minutes.

MIDlet suites are uninstalled and their RMS record stores, if present, are deleted if any of the following events occur:

- If Extension Mobility is used to change the current active user of the phone (occurs in login and logout)
- If the phone registers to a different Unified CM cluster which does not

support MIDlets or has a different services configuration for the device

- If nobody is logged into the phone via Extension Mobility, but the static "owner" of the phone is changed in Unified CM Admin configuration (this changes the current active user of the phone)
- If the configuration is cleared from the phone using the Settings menu or by any other means (such as a factory reset).

11.2 Quickstart Options for MIDlets

The 7925G and 7926G Cisco Unified Wireless IP phones have some unique mechanisms for invoking a Java MIDlet quickly, and can also be used as a way to bootstrap or auto-load a Java MIDlet.

The 7925G and 7926G Cisco Unified Wireless IP phones possess a button on the left side of the phone, often referred to as a PTT button. This button can be used to invoke an XML service, and the service is defined as the "Application URL"within the Product Specific Configuration Layout section for the7925G and 7926G Cisco Unified Wireless IP phones within the Cisco Unified Communication Manager.

If the "Application URL" defined in this field, is the same URL as the URL that is listed for the first Java MIDlet in the list of Java MIDlet services that the phone is subscribed to, pressing the PTT button will attempt to load the URL and start the Java MIDlet. This is a convenient way to bring a MIDlet that is in a paused state, or background state, back into foreground with the press of a single button.

A similar mechanism is available through the "Idle" URL. If the "Idle" field in the External Data Locations Information contains a URL that is the same location URL as the first Java MIDlet in the list of Java MIDlets that the phone is subscribed to. If the phone is idle, when the idle timer expires, the phone will try to launch the Java MIDlet and bring it to foreground. This is a convenient way to "auto-start" a Java MIDlet with no user intervention.

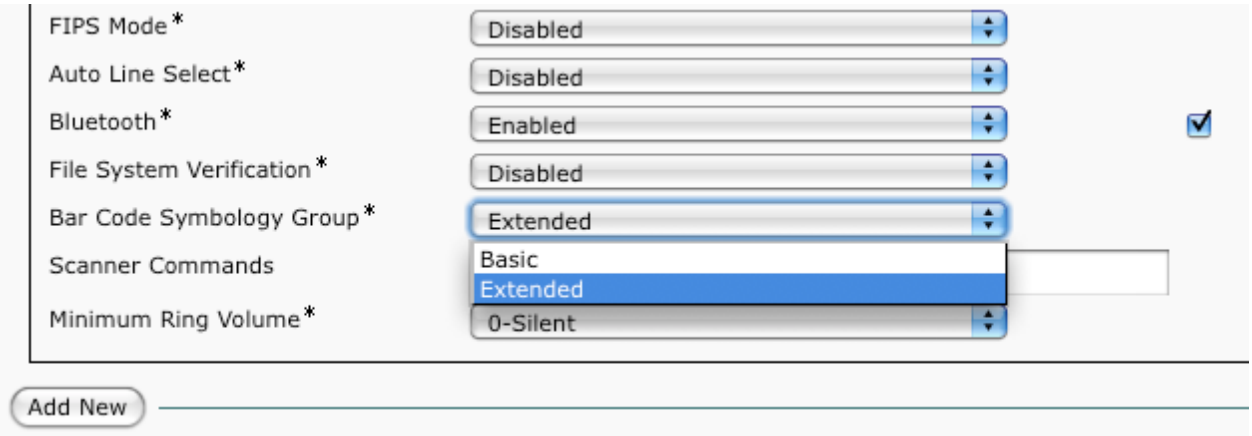
It should be noted that the 7925G and 7926G Cisco Unified Wireless IP phones will attempt to re-launch the Java MIDlet each time the idle timer expires. The idle timer will not start if the phone is engaged in a phone call, or if the user is interacting with the phone. The idle timer will only start when the phone is truly idle.

11.3 Bar Code Scanner provisioning

Bar code scanners have a range of different bar code types that can be read, these are know as Symbologies. The Cisco 7926 can read a variety of symbologies, and the number of symbologies the scanner is configured to recognize can have an impact on the time taken to recognize a bar code.

In an attempt to simply the provisioning, the bar code symbologis have been put into two groups, and the provisioning of these groups is defined through the device specific configuration page for the 7926.

The two groups are Basic and Extended



The bar code symbologies read by the respective groups is listed in the following table

Basic	Extended
Code39	Code39
Code128	Code128
DataMatrix	DataMatrix
EAN13	EAN13
UCC/EAN128	UCC/EAN128
UPC	UPC
PDF417	PDF417

Aztec
Codabar
Code11
Code93
EAN add on 2
Interleave 2 of 5
Matrix 2 of 5
Plessey
GS1 Databar
Standard 2 of 5
Telepen
QRCode
Maxicode
MicroPDF417

In addition to these symbology definitions, the scan engine can be further configured using a scanner command protocol. These command protocol is used to define some of the pattern matching characteristics of the bar code scanner.

The Intermec Scanner Command Protocol (ISCP) consists of hexadecimal strings sent to the scanner via a serial link, in a similar fashion to modem AT commands.. If a developer does know which ISCP commands they wish the bar code scanner to read in during initialization, these Hexadecimal values can be entered in the "Scanner Commands" field of the device specific configuration pages for the 7926G. If more information is desired about the ISCP command strings available, and their application, please contact Cisco.

Should a set of commands cause issues, and some confusion arise as to the state of the scanner, the scan engine can be brought back to its initial state through the 7926G's local settings page. Select Settings -> Phone Settings -> Diagnostics -> Scanner the options to carry out a bar code scan (without requiring a Java MIDlet application), or to reset the scanner, are available.

11.4 Web Server Settings for Installing MIDlet Applications

By default, some web servers are not configured to recognize JAD files. Before you install MIDlet applications, you must make the following changes to the web server to ensure that it recognizes JAD and JAR file types:

IIS Web Server

- For JAD File,

  - set the File extension to .jad
  - set the MIME type to text/vnd.sun.j2me.app-descriptor
- For the JAR file,
  - set the file extension to .jar
  - set the MIME type to application/java-archive

For more information on configuring MIME types on IIS, refer to the following links:

IIS 6:
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/cd72c0dc-c5b8-42e4- 96c2-b3c656f99ead.mspx?mfr=true

IIS 7: http://technet.microsoft.com/en-us/library/cc753281%28WS.10%29.aspx

Apache Web Server

Add the following lines in the "mime.types" configuration file:

"text/vnd.sun.j2me.app-descriptor jad"
"application/java-archive jar"

12 Cisco Extensions to MIDP

12.1 Overview

The MIDP extensions provide additional classes for customizing user experience and enabling device specific capabilities. As a MIDlet developer, you can use these extensions to write applications tailored to the Cisco Unified IP Phones. This chapter provides a high level overview of the extensions for Wireless IP Phones.

12.2 Scanner APIs

Table 8-2 lists the capabilities of ScannerAPI class used for Cisco Unified Wireless IP Phone 7926G. A 7925G has the same functions defined, but only the isScanCapable() function is implemented. The other functions do nothing on the 7925G.

Table 8-2 Methods in ScannerAPI

Methods in ScannerAPI	Method Description
boolean isScanCapable()	7925G: Returns false. 7926G: Returns true.
addListener(ScanListener listener)	Registers the class as a listener for scan results. Calling this method replaces any prior listeners on the same ScannerAPI.
removeListener(ScanListener listener)	Clears the listener from receiving scan results.
int start()	Initiates a scanning event. The barcode reader will light. Returns a status code. (Codes listed below)
int stop()	Aborts the scanning that is in progress. Returns a status code. (Codes listed below)

In order to receive notifications when barcodes are returned from the ScannerAPI, a MIDlet must have a class implementing the ScanListener interface. This class must be registered with the ScannerAPI using addListener(theClass). Table 8-3 lists the abstract function in the ScanListener interface.

Table 8-3 Methods in ScanListener

Methods in ScanListener	Method Description
void scanResult(String barcode, int status)	This method should be implemented by any class wishing to receive notifications when the barcode scanning action completes. The two parameters are the scanned barcode data and the status code of the scanner. (Codes listed below)

Table 8-4 lists the status codes delivered by the ScannerAPI.

Table 8-4 Scan Status Codes

Status Codes	Status Description
SCAN_STATUS_OK	ScannerAPI is behaving normally.
SCAN_STATUS_INPROGRESS	ScannerAPI is currently scanning a barcode.

SCAN_STATUS_FAIL	ScannerAPI failed to retrieve a barcode.
SCAN_STATUS_TIMEOUT	ScannerAPI failed to retrieve a barcode in defined timeout.

13 Best Practices –

13.1 Graphical User Interface Objects

The MIDP LCDUI API provides the basic GUI components for building common applications (Lists, Alerts, Forms and Items, and so forth). The user interaction of those standard GUI components has been designed and approved by the Cisco User Experience team to provide an optimized user experience which consistent throughout all phone applications. To improve the usability of MIDlet applications, developers of custom user interface components should consider how well those components adhere to the overall user experience of the Cisco Unified IP Phone.

The following list provides general guidelines for developing the look (rendering style and appearance) and feel (user input and interaction) behavior of custom UI components such as CustomItems, Canvases, and GameCanvases:

- Make sure you fully understand the capabilities and usage of the standard UI components and use them whenever possible. Custom components not only cost in initial development, but also typically require porting effort when moving to different devices/models.
- When rendering custom components, use the standard system color palettes retrieved from the Display.getColor(int colorSpecifier) method to ensure the color scheme matches the standard UI components and is optimized for the hardware display of that particular phone model. This minimizes your porting efforts to new devices/models.
- The Canvas and CustomItem APIs for querying for device input capabilities can be useful in some cases, but be careful not to create drastically different user experiences on different phone models because it makes it difficult for the user to switch between them. For example, the basic user interaction should work with 2-way navigation and softkeys to ensure support on all models, and 4/5-way navigation and touch support should augment/optimize/accelerate that user interaction on the higher-end models.
- Touching of items within a custom selection component (list, grid, and so forth) should not

only result in moving the focus/highlight, but also be the default invocation action.

- The **Select** key, if present, performs the same action as touching a highlighted item. Both these methods invoke a default action.

- For CustomItems which present some "editable" entity, consider specifying line breaks before and after the item, and using left/right navigation for modifying the value of the item. Vertical navigation should be used to navigate into and out of the item. Left/right Commands (softkeys) should also be presented as Item Commands to allow the value to be changed on phone models which do not support 4/5-way navigation.
    - For CustomItems, consider using similar visual traits, such as borders and highlight bounds, as the standard UI components.

13.2 Memory and Resource Optimization

The current MIDlet container running in Cisco Unified IP Phones is classloader-based and efficiently enforces security policies to prevent access to unauthorized data. However, the MIDlet container cannot completely isolate applications from a resource-consumption perspective (heap, threads, so forth). Therefore, it is absolutely critical that you test all MIDlets running on the phone to ensure that they are free of memory leaks, and that they cleanly shutdown any threads or timers that they create. Failure to do so degrades performance and eventually cause the phone to reset, or require the phone to be reset before running another MIDlet application.

13.3 Free Memory Allocation Examples

Cisco Unified IP Phones are embedded systems, and as such the memory utilization of these devices is dependant on the configuration and operating parameters in use. Table 9-1 provides guidance about the free memory that is available for Java MIDlet applications. It lists free memory allocation examples that can be used when using a Cisco Unified IP Phone 7975G running firmware release 8.4(3) or later and operating with the SCCP protocol.

13.4 Java Heap (RAM)

Approximately 2 MB of memory is available for use by Java MIDlet applications—500 KB of Java Heap space (for Java object creation) and 1.5MB of free JVM memory (for MIDlet class loading, graphics resources, and so forth.). The exact amount of free memory available varies by phone model and the exact state and configuration of the phone. Therefore, it is critical to test your Java MIDlet application to ensure it stays within the specified constraints. For more information on measuring the amount of memory consumed by the Java MIDlet application, see the "Analyzing Memory Usage" section on page 10-2.

There are many ways to optimize the memory footprint of Java applications, most of which are not specific to Cisco Unified IP Phones. However, because limiting memory consumption on Cisco Unified IP Phones is critical for ensuring stability, there are some key points to remember when designing and implementing your applications:

- Exit Threads as soon as possible, and verify that all Threads and Timers are cleanly terminated when the MIDlet exits. Otherwise, the MIDlet will not be allowed to start again until the phone is reset because the phone will assume that the MIDlet is still running.

- Nullify references to objects as soon as they are no longer used—especially for objects which consume significant native resources such as Thread, Timer, Image, Font, Canvas, and GameCanvas.
- Use static fields and collections classes with caution to ensure that objects are not left lingering in memory because of unnecessary references.

The table below describes the recommended maximum memory usage of various Cisco Unified IP Phone models.

Table 9-1 Recommended Memory Usage

Phone Model Number	**Recommended Maximum Memory Usage**
7925G, 7926	1 MB

13.5 Persistent Flash Storage

Exactly 2 MB of total flash space is available for all Java MIDlet applications. This includes the storage space for the MIDlet applications themselves (JAD and JAR files), as well as any RMS RecordStores that the MIDlets might create. Any individual MIDlet JAR file is also limited to 512 KB. These limits are strictly enforced by the phone and if they are exceeded, the MIDlets will fail to install and/or RecordStore operations will throw runtime exceptions and fail.

13.6 Threads and Timers

As Java MIDlet applications run, they are free to create their own worker Threads for handling

network I/O, animation loops, Timers, and so forth. The MIDlet container inside the Cisco Unified IP Phone tracks all Threads that are created by the MIDlet as it runs and ensures that all of the Threads terminate when the MIDlet closes. If any Threads are left running when the MIDlet exits, the following occurs:

- The phone reports a generic "Error, contact administrator" error message to the user to indicate that an error occurred while trying to close the application.
- A detailed error message and a list of still-running Threads, is logged in the phone console logs for debugging purposes.
- No MIDlets are allowed to run again until the phone is reset. This guarantees that the MIDlet Thread was terminated and that it cannot cause any security or stability problems.

All MIDlet applications must ensure that all of the Threads created by the MIDlet are cleanly terminated when it exists, and that all of the Timers are cancelled (which will internally exit the Timer's Thread).

13.7 CPU Utilization Monitor

MIDlets can create worker Threads for handling asynchronous tasks such network I/O, animations, and so forth. However, the phone firmware also implements an internal monitoring mechanism which monitors CPU utilization and resets the phone if a specific threshold is crossed. This protects the phone in the case of a software failure in which the CPU is maxed for an extended period of time. Therefore, when you use Threads that are for long-running tasks such as animations, you must be careful not to allow the MIDlet to reset the phone.

It is difficult to estimate how much CPU is available to the MIDlet because CPU estimations are dependent on how many other tasks or running on the phone. However, you should refer to these general guidelines:

- When testing your application, load the phone with as many other tasks as would typically be seen in production—for example, multiple concurrent calls with one of the active.
- Monitor the console logs and look for MAX_CPU warnings. These are generated for every CPU second which exceeds the threshold. An occasional warning is to be expected, but if you see them for several consecutive seconds, you should investigate how to reduce CPU utilization. In the case of animations, reducing the frame rate or the animation size is the best approach.
- If you have other long-running tasks which are not timer-based (like animations), then structure your code such that the worker Thread can be paused on a regular basis to avoid MAX_CPU resets.

It is recommended that there is a pause 100ms out of every second. This not only prevents a MAX_CPU reset, but also improves UI responsiveness of the application.

13.8 Graphics Resources

The primary purpose of most Java MIDlet applications is to provide a robust, compelling user interface on embedded devices, and as such, they tend to consume a large amount of memory for graphics-related resources (Image, Font, Canvas, GameCanvas, etc.).

It is critical to only create them when necessary and to release them as soon as possible. For more information, see the "Java Heap (RAM)" section on page 9-2. Some additional tips for handling graphics resources are as follows:

- When creating PNG image resources, follow the guidelines in the "Image Formats PNG-8 vs. PNG-24" section on page 9-5 for choosing the optimum image format.
- Share and reuse objects whenever possible. Only create one Canvas or GameCanvas and share it across all the various screens of the application. Do not create multiple Image objects that are loaded from the same PNG file. Also create as few mutable Images as possible and share them throughout the application.
- To minimize fragmentation of graphics memory, create large resources first whenever possible. For example, create your Canvas or GameCanvas at startup, and create large Images (mutable or immutable) first, especially if they will be long-lived. Load smaller images and icons last or dynamically as needed.

13.9 Image Formats PNG-8 vs. PNG-24

There are two PNG image formats used for Java MIDlet resources—PNG-8 (palletized) and PNG-24 (ARGB). Table 9-2 lists the relative pros and cons, including recommended usage:

Table 9-2 Image Format Comparison

Type	Pros	Cons	Suggested Usage
PNG-8 (palletized)	1 byte per pixel, and very efficient for large images	• Image limited to 256 unique palette colors, therefore not recommended for photographic or gradient	Images larger than normal icon size (approximately 20 x 20) should use palletized images to conserve memory.

		images • Palette itself consumes at static 1KB (256 colors x 4 bytes per color) • Does not support semi-transparency and all pixels must be fully opaque or fully transparent	
PNG-24 (ARGB)	• Supports as many unique colors as the phone hardware is capable of rendering • Supports 256 levels of per-pixel semi-transparency (alpha channel) • No color palette used so no 1KB of overhead on small images	Consumes 2-4 bytes per pixel depending on phone model, and is costly for large images	Typical icon-size images (approximately 20 x 20). Also used when semi-transparency or greater color counts is required, but use cautiously to avoid running out of memory.

13.10 Multi-Threaded Applications

When a Java MIDlet is started, it gets its own message-queue (MQ) Thread. This Thread is used to deliver all callback notifications to the Java MIDlet, including start/terminate requests, user I/O, and focus change events. This follows a very common GUI development model where a single “event thread” is used to deliver events to the MIDlet. For basic MIDlet applications, this MQ thread may be all that is required and, therefore, the MIDlet does not perform any multi-threading.

However, most Java MIDlet applications employ some form of multi-threading, with the most common use-cases being network listener/receiver Threads and Timers. As per MIDP API specification, nearly the entire MIDP LCDUI API is completely thread-safe which means that any of the API methods can be called by not only the MQ thread, but also by any other thread that the MIDlet might have created, including worker threads for background processing or network I/O, or Timer callback events. This is highly unusual for a GUI library because most others will specifically state that they are not thread-safe at all (with the exception of a few methods that are used to post events into the event thread for processing).

If the MIDP LCDUI API is thread-safe, the rest of the code in the MIDlet itself will not be used unless it is carefully designed and implemented as such. Therefore, the best programming approach is to immediately dispatch any events received from other threads into the event thread for processing. This makes your application single-threaded so that other more complex synchronization mechanisms are not required. Several examples of dispatching events to the MQ thread are provided in the SDK, but the following example illustrates the approach:

```
public class MyDatagramReceiver extends Thread {

    ..... other attributes, methods, and initialization stuff

    public void run() { while
        (!stopped) {
            try {
                final Datagram dgram =
                msgConn.newDatagram(MAX_DGRAM_SIZE);
                msgConn.receive(dgram);
                System.out.println("#### DeviceManager.onRun() B");
                theDisplay.callSerially(new Runnable() {
                    public void run() {
                        datagramReceived(dgram);
                    }
                });
            }
            catch (Exception ex) { ex.printStackTrace(); }
        }
    }
    private void datagramReceived(Datagram dgram) {
        //  Do whatever processing needs to be done for this datagram
        //  Note that this method will only be called the MQ thread – never
        //  by this MyDatagramReceiver thread
    }
}
```

In the example above, the network I/O thread (MyDatagramReceiver) is the thread that blocks and listens for an incoming datagram. It then reads the datagram bytes from the network and puts them in the datagram, but is not the thread that calls the datagramReceived() method. The Display.callSerially() dispatches a message to the MIDlet's MQ thread to instruct it to execute the specified Runnable object as soon as it becomes available. Then the MQ thread calls the datagramReceived() method. The benefit of this approach is that the datagramReceived() method is then free to call any methods on any objects because there are no concerns about thread-safety. From the MIDlet's perspective, the application is single-threaded.

13.11 Internationalization and Localization

Unlike the more robust JavaSE APIs, the MIDP API does not provide utility classes for bundling and accessing locale-specific resources such as text phrases and graphic resources. The microedition.locale System property provides the current user locale setting and that provides basis for building such a localization mechanism as described in “System Propetties”.

Most Java MIDP IDE’s have some level of built-in internationalization support to make up for the inherent shortcomings in the MIDP API spec. NetBeans, for example, has a built-in Internationalization Wizard which generates the necessary code to support basis phrase localization, as well as managing the resource files themselves. Therefore, if this meets the application requirements, then this is the simplest approach. NetBeans generates standard Java .properties files that contain the phrases for each locale and also provides a simple GUI interface for managing the text phrases for each locale.

13.12 Command Types and Positioning

When using Commands (softkeys) in Java MIDlets, the application should always use an appropriate Command Type to ensure that the Command gets positioned in an appropriate softkey location for the given device. Table 9-3 criteria is used by the Cisco Unified IP Phones for positioning Commands.

Table 9-3 Criteria for Positioning Commands

Command Position	**Command Types (in order of preference)**
1	Command.OK, Command.ITEM
2	Command.STOP
3	Command.CANCEL, Command.EXIT, Command.BACK
4	Command.SCREEN
5	Command.HELP

Cisco Unified IP Phones listed in Table 1-1, allocate softkey positions in the following order:

1. System Features

Any “system” softkeys that are required by the implementation of the current Displayable object are assigned first. This includes softkeys for actions like the default Select/Deselect for Exclusive and Multiple Lists, Backspace for TextBoxes, and so forth. These are the softkeys that

are provided by the platform for interaction with the object which are outside of the control of the MIDlet.

2. Displayable and Item Commands

After positioning the System Features, the phone iterates through each softkey position (1-5), and searches for the first Command which matches that type. Commands are matched against types in the order shown in Table 9-3. For example, the phone searches for OK Commands before ITEM Commands when populating position 1. Displayable Commands are searched before Item Commands. If after iterating through all softkey positions, Commands remain which cannot be placed into their preferred position (because it is already occupied), then they are assigned to the first available softkey position. Displayable Commands are placed first, then Item Commands.

The following are examples of the rules defined above:

Example 1

Form myForm = new Form("My Form");

myForm.addCommand(new Command("Exit"), Command.EXIT, 0));

myForm.addCommand(new Command("Ok"), Command.OK, 0));

Ok		Exit	

Example 2

Form myForm = new Form("My Form");

myForm.addCommand(new Command("A"), Command.OK,

0)); myForm.addCommand(new Command("B"),

Command.OK, 0)); myForm.addCommand(new

Command("C"), Command.OK, 0));

myForm.addCommand(new Command("D"), Command.OK,

0));

A	B	C	D

Example 3

Form myForm = new Form("My Form");

myForm.addCommand(new Command("A"), Command.STOP,

0)); myForm.addCommand(new Command("B"),

Command.STOP, 0)); myForm.addCommand(new

Command(“C”), Command.STOP, 0));

myForm.addCommand(new Command(“D”), Command.STOP,

0));

B	A	C	D

Example 4

Form myForm = new Form(“My Form”); StringItem

myItem = new StringItem(“My Item”);

myItem.addCommand(“ItemOk”, Command.OK, 0);

myItem.addCommand(“Stop”, Command.STOP, 0);

myForm.append(myItem);

myForm.addCommand(new Command(“FormOk”), Command.OK,

0)); myForm.addCommand(new Command(“Select”),

Command.ITEM, 0));

FormOK	Stop	Select	ItemOK

Note Cisco Unified IP Phones do not currently use the Command Priority attribute when determining softkey placement.

13.13 RecordStore and RecordEnumeration Performance

The MIDP Record Management System (RMS) enables MIDlet applications to read and write application-specific information to the system storage of the device, such that it can be retained even after multiple MIDlet startups. This information is not lost when the device power is reset.

Depending on the application use case, you can use one of the following methods to retrieve records that have been written to a RecordStore:

- Record ID number—Enables an application to use specific record ID numbers to retrieve a record. This method is useful when smaller numbers of records are to be retrieved from the RecordStore. In this case, the RecordStore.getRecord (int recordID) method is used.
- RecordEnumeration—Enables an application to retrieve a large number of records and is desirable when the order of retrieval is not critical. This method provides better performance than using specific ID numbers because the platform can directly read them from the file

system in the order in which they are stored. This method does not require searching, indexing, or caching, which is required when locating a specific record based on the record ID. For example, if a MIDlet application uses an RMS RecordStore to persist localization phrases that are read from the RecordStore upon MIDlet startup, then creating a RecordEnumeration and calling RecordEnumeration.nextRecord() improves performance significantly.

Note Using the RecordComparator to sort the RecordEnumeration degrades read performance because the records must be read and sorted very quickly.

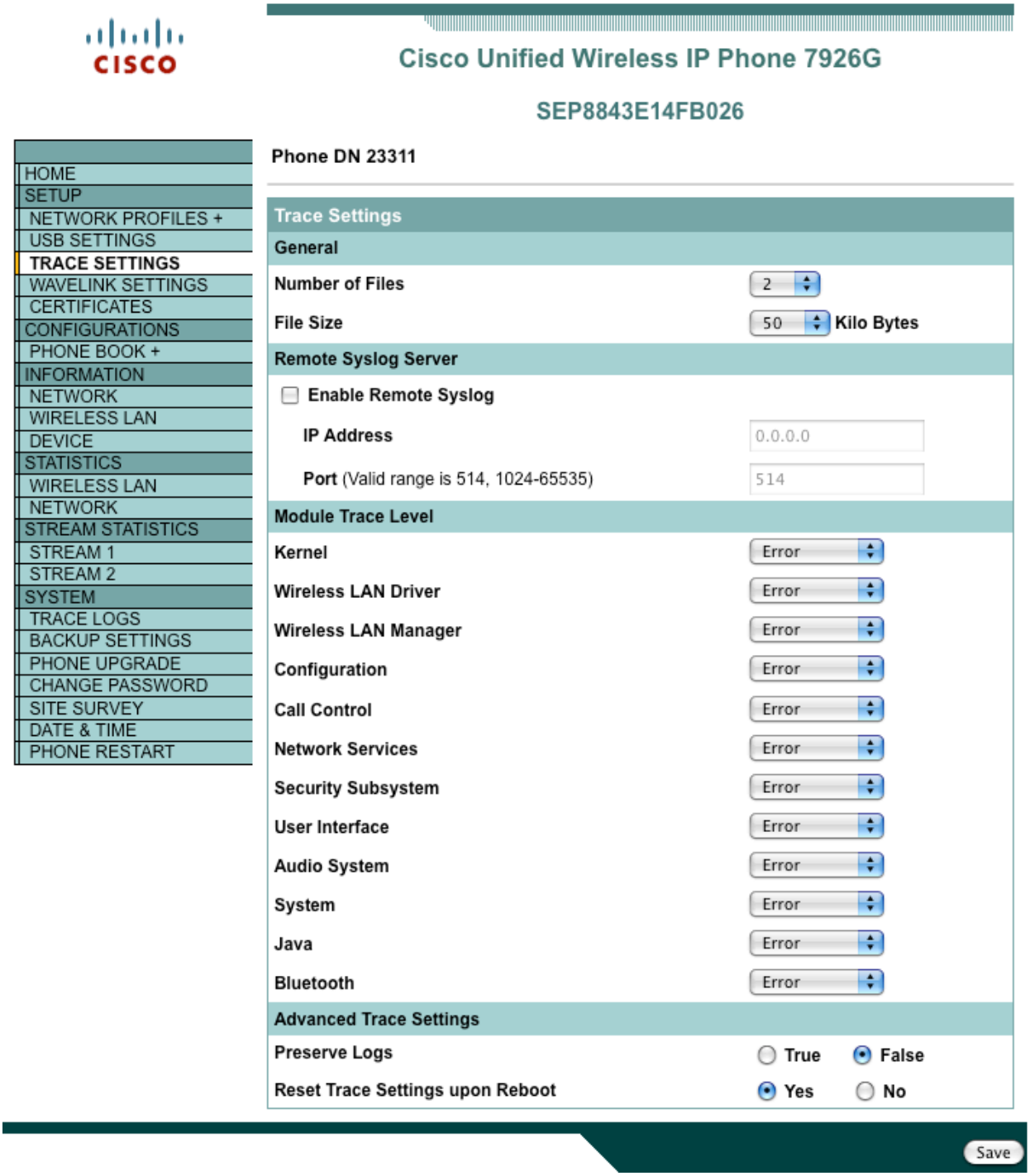
14 Troubleshooting –

14.1 System Output for Development Debugging – Handheld phones

The 792x series of phones utilize a different management interface for remote access to the phones and for collecting troubleshooting data.

The 792x hand held phones have a web page, with contains the ability to set trace files, collect trace files, and configure some parameters. Admin access to the web page is controlled by the settings in the CUCM under the product specific configuration page. Once admin settings and web access has been enabled, further configuration can take place on the phone.

The Java module would prove the most useful module for debugging any MIDlets.



Trace Modules

Kernel	Operating System
Wireless LAN Driver	Channel scanning, roaming, authentication
Wireless LAN Manager	WLAN Management, QoS
Configuration	Phone configuration, firmware upgrade

Call Control (SCCP)	Cisco Unified Communications Manager messaging
Network Services	DHCP, TFTP, CDP, WWW, Syslog
Security Subsystem	Application level security
User Interface	Keypad, softkeys, MMI
Audio System	RTP, SRTP, RTCP, DSP
System	Event Manager
Java	Java Virtual Machine
Bluetooth	Bluetooth

Trace Levels

Various levels of tracing are available which provide different levels of messaging.
Emergency, Alert, Critical, Error, Warning, Notice, Info, Debug

Note: All trace modules are set to Error level by default.
Voice quality can potentially be impacted if higher trace levels are configured or if **"Preserve Logs"** is enabled, which will write the logs to flash memory.
The trace level will reset to **"Error"** level by default unless configured to preserve the trace levels.

Trace Files

By default the trace files are only held in memory and will be deleted when the phone is reset. Setting the value "Preserve Logs" will ensure that that trace files are written to flash memory and preserved after a reset. The number of files, and size of the files, can be managed by the web interface. Writing the trace files to flash memory takes a little longer than storing the trace files in DRAM, and this may some effect on voice quality.

14.2 Java Heap (500 KB for MIDlets) and Creating Java Objects

The Java Heap is a fixed-size, pre-allocated block of memory which is used by the JVM for creating Java objects. The size of the Java Heap and the amount free heap available varies by phone model and configuration, but 500 KB has been reserved for use by Java MIDlets. Therefore, you should stay within that guideline.

You can view the size and free statistics of the Java Heap by using the Memory Status Tool on the phone web page.

The Total Heap Size and Free Heap Size are expressed in bytes. So in the previous example, the total amount of Java Heap in use is 2,000,384 minus 853,800, which equals 1,146,584 bytes.

14.3 Dynamic Memory Allocation (1500KB for MIDlets)

The JVM requests additional memory as needed to perform internal tasks such as opening JAR files and loading classes and resource files. Other native tasks, such as the 2D graphics engine will also dynamically allocate additional memory from the OS as it perform tasks, such as creating Images and Fonts. In order to accurately measure the total amount of memory consumed by the MIDlet at runtime, you must measure the amount of Java Heap consumed by the MIDlet, as well as the increase in the amount of dynamically allocated memory. For most Java MIDlet applications, the amount of dynamic memory consumed is several times larger than the amount of Java Heap consumed. The size of the Dynamically Allocated Memory and the amount free varies by phone model and configuration, but 1500 KB has been reserved for use by Java MIDlets. Therefore, should make sure that you stay within that guideline.

You can view the size and free statistics of the Dynamic Memory Allocation by using the Memory Status Tool on the phone web page.

The total amount of Dynamically Allocated Memory for the JVM process is given by the usmblks value (8,708,528 bytes), and the portion of that memory which is currently free (not in use) is given by the fordblks value (190,864 bytes). In this example, the total Dynamically Allocated Memory in use by the JVM is 8,708,528 minus 190,864, or 8,517,664 bytes.

Note Memory statistics are reported in number of blocks. The operating system block size is 1 byte, therefore 1 block is equal to 1 byte.

14.4 Measuring Total MIDlet Memory Usage

To measure the total amount of memory consumed by a Java MIDlet, you must capture memory usage snapshots for Java Heap and Dynamic Memory Allocation before the MIDlet is started (preferably just after a phone reset) and again after the MIDlet is running. The difference between the two snapshots is considered to be the total memory consumed by the MIDlet at runtime.

To capture memory usage snapshots, and also sample output and calculations, perform these steps:

Procedure

Step 1 Perform a hard reset on the phone and wait for it to register with Cisco Unified Communications Manager and become idle.

Step 2 Browse to the phone web pages and record the Memory Status Tool values.

Step 3 Start the Java MIDlet application and exercise its functionality as much as possible to bring the application into a state of peak memory consumption.

Step 4 Browse to the phone web pages again and record the Memory Status Tool values.

Step 5 Subtract the free memory value recorded in Step 4 from the free memory value recorded in Step 2 to calculate the total Java Heap and Dynamic Memory Allocation (System Free Memory plus Java Pool Free Memory) used by the MIDlet.

Note Make sure the amount of Java Heap does not exceed 500 KB and the amount of Dynamic Memory Allocation does not exceed 1500 KB. If they do, the MIDlet will not be approved for execution on Cisco Unified IP Phones. Also, it will not pass IVT testing and get Cisco signed.

14.5 Finding Memory Leaks

If the total memory usage of the Java MIDlet continues to increase unexpectedly as the MIDlet runs, there might be a memory leak in the MIDlet code. For more information on calculating memory usage, see the "Measuring Total MIDlet Memory Usage" section on page 10-3.

Memory leaks in Java applications are caused by the creation of Java objects which are not freed in a timely manner so that the memory can be reclaimed by the JVM Garbage Collector (GC). Typical causes for memory leaks are lingering references to objects in collections classes (such as Vectors and Hashmaps). For example, registering an object as a listener and then forgetting to unregister it once the object is no longer needed. The collections class that is holding the list of listeners continues to reference the object, which prevents it from being reclaimed by the GC.

Java memory leaks are typically found by using a Java profiler attached to the running JVM. The profiler allows you to monitor all of the Java objects that are created, where they are created, and what other objects are holding references to those objects and preventing them from getting garbage collected. This is how it works on the Cisco Unified IP SoftPhone since it runs on a PC using a standard JavaSE VM which supports attachment of a profiler. However, a Cisco Unified IP Phone does not support the attachment of a profiler, so memory leaks are typically debugged on the Cisco Unified IP Phone Simulator.

14.6 Finding Memory Leaks Using the Cisco Unified IP Phone Simulator

14.7 Find Out What Objects Are Leaking

As the JVM runs, the Cisco Unified IP Phone Simulator allows you to take snapshots of the Java Heap at different points in time and then run the profiler to generate different reports that show which objects are increasing in quantity from one snapshot to the next. This allows you to identify what objects are leaking.

14.8 Find Out Where Leaking Objects Are Being Created

The Cisco Unified IP Phone Simulator allows you to drill down into specific objects and locate exactly where in the code the object was created.

14.9 Find Out Why Leaking Objects Are not Being Freed

The Cisco Unified IP Phone Simulator allows you to dump portions of the reference tree that shows all of the references between objects. By inspecting these references, you can see what other objects are referring to those leaking objects and preventing them from getting garbage collected. Once you know where the unwanted reference exits (for example a Vector or Hashmap), you can correct your code to make sure that reference gets removed once it is no longer needed.

Many profilers, such as JProbe, also include “leak detector” features would simplify the task of finding memory leaks by allowing you to select a specific object, and then build a tree of dependencies from that object. You can also simulate the result of removing a specific reference to see if doing so will free all references and allow the object to be garbage collected.

14.10 Memory Status

The Memory Status feature enables administrators and third-party developers to analyze Cisco Unified IP Phone memory use and also prevent memory-related issues before they launch applications.

Administrators can use the Memory Status feature to identify the root cause of insufficient memory-related crashes even after the phone is reset. Third-party developers can also use the Memory Status feature to determine how much memory specific applications, such as XSI and MIDlets use.

The Memory Status feature can be accessed from the phone web page and users can view the following memory-related information from the Device Information window:

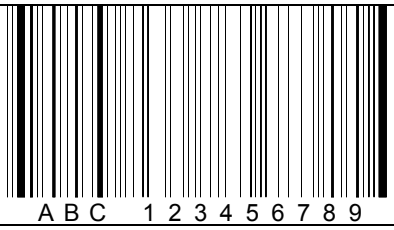
- System Free Memory
- Java Heap Free Memory
- Java Pool Free Memory

The memory status is displayed when the free system memory is below 1MB. The system displays the memory status on any subsequent memory request. Log messages are stored in the

system log and administrators can access it from the phone web page.

14.11 Barcode Samples

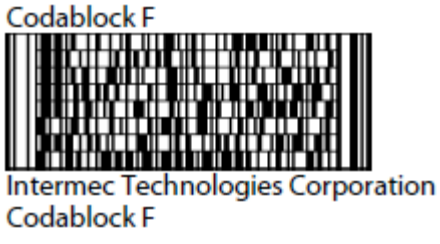
In the following table, barcodes marked with (B) belong to Basic Group, barcodes marked with (E) belong to Extended Group. Barcodes marked with (X) are not supported.

Subject	Test items	Barcode example
1 - D	**Enable Codabar (E)**	
	Enable Code 3 of 9 (B-Code39)	
	Enable Code 11 (E)	
	Enable Code 32 Pharmaceutical(PARAF) (X)	
	Enable Code 93 (E)	
	Enable Code 128 (B)	

	Enable EAN with Add-On EAN with Extended Coupon Code (X)	
	Enable EAN-13 (B)	
	Enable Interleaved 2 or 5 (E)	

Subject	Test items	Barcode example
1 - D	**Enable Matrix 2 of 5 (E)**	
	Enable Plessey (E)	
	Enable GS1DataBar (E)	
	Enable Standard 2 of 5 (E)	
	Enable Telepen (E)	
	Enable Trioptic	

	Code (X)	
	Enable UCC/EAN 128 (B)	
	Enable UPC (B)	
	Enable Straight 2 of 5 IATA (X)	

Subject	Test items	Barcode example
1 - D	**Enable UPC-A (B)**	
	Enable Straight 2 of 5 Industrial	
2 - D	**Enable MicroPDF (E)**	
	Enable Codeblock F (X)	
	Enable Aztec	

	(E)	
	Enable QRCode (E)	 ABC 123456789
	Enable MaxiCode (E)	 ABC123456789
	Enable DataMartix (E)	 ABC123456789
	Enable PDF417 (B)	 **123456789**
	EAN13 Add On 2 (E)	 5 012345 678900 **50123456789012**