

SCMAD Exam Notes

Objectives 8 (MIDP UI API) and 9(MIDP GAME API) are not included.

by
Eduardo Rodrigues

Any comments please, mail to duardor@terra.com.br.

Sorry for my mistakes with English...I'm improving it day after day (I hope so! ☐)

Section 1: JTWI (JSR 185) and Overview / JTWI-compliant Wireless Applications

Good Articles:

Future Java Technology for the Wireless Services Industry -
<http://developers.sun.com/techtopics/mobility/midp/articles/j2mefuture/>

1.1 - Identify the goals and characteristics of the JTWI specification (JSR 185), including the mandatory specifications, conditionally required specifications, and the minimum configuration. Compare the relationship and differences between JTWI and other wireless Java technologies.

[JTWI] 1.2 - > Goals

Minimize API fragmentation in the mobile phone device market. Improve compatibility, interoperability and completeness of J2ME technology.

Three ways to achieve this goal:

- 1 - Specifies a common set of APIs by requiring the use of a number of component specifications.
- 2 - Mandates elements of the component specifications, eliminating optional features.
- 3 - Clarifying some elements of the component specifications.

[JTWI] 1.3.1 - > Mandatory specification

MUST implement each of the mandatory specifications.

MIDP 2.0

- Record Store Minimum -> create at least 5 record stores.
- HTTP Support for Media Content -> must support HTTP 1.1.
- JPEG for image objects -> Only required format is PNG.
- Timer resolution -> Must permit application set values for firstTime, delay and period parameters for java.util.timer.schedule(). No more than 40 ms.
- Minimum number of timers -> create a minimum of 5 simultaneously timers.
- Bitmap minimums -> Must support PNG images with pixel color depths 1,2,4,8,16,24 and 32 bits per pixel. Support images up to 32768 pixels.
- Text field and text-box coupling phone book -> Must implement a mechanism to select data from phone-book when editing TField or TBox.
- Supported characters in TextField and TextBox->Must support a set of special characters on editing TField or TBox. See Table 4 of the MIDP spec.
- Supported character in EMAILADDR and URL Fields -> Must support the same special character of table 4. See above.
- Push registry alarm events-> Must implement alarm-based push registry. If no security mechanism is present, the PushRegistry alarm must get explicit permission from the user.
- Identification of JTWI via System Property -> Value of the system property microedition.jtwi.version must be 1.0.

WMA1.1

- Support for SMS in GSM Devices -> Must support GSM SMS service using the api describe in appendix A of the WMA spec.
- Cell Broadcast in GSM devices -> Must follow spec WMA appendix B.
- SMS Push -> Must support MIDP 2.0 push handling. Follow the WMA spec appendix D. If no other security mechanism is present must get user permission.

[JTWI] 1.3.2 - > Conditionally required specification

MMAPI 1.1

HTTP 1.1 Protocol -> HTTP 1.1 must be supported for media file download for all supported media formats.

MIDI Feature Set -> must implement the MIDI feature / playback.

Controls for MIDI Feature Set -> Support for VolumeControl must be implemented.

Tone Sequence File Format -> Tone sequence file format must be supported.

JPEG Encoding in Video Snapshots -> a implementation that supports the video feature set and video image capture must support JPEG encoding.

[JTWI] 1.3.3 - > Minimum configuration

CLDC 1.0 is the minimum required J2ME configuration.

If floating point capabilities are exposed to Java applications, CLDC 1.1 or another compatible J2ME configuration must be implemented.

Minimum Application Thread Count -> Must allow to create a minimum of 10 simultaneously running threads.

Minimum Clock Resolution -> The method `java.lang.System.currentTimeMillis()` must record the elapsed time in increments not to exceed 40 ms.

Custom Time Zone ID -> Must permit the use of custom time zones.

Names for Encodings -> Must support at least the preferred MIME name.

Character Properties -> Must provide support for character properties and case conversions for characters in the "Basic Latin" and "Latin-1 Supplement" blocks of Unicode 3.0

Unicode Version -> Must support Unicode characters. Character information is based on the Unicode Standard, version 3.0.

1.2 - Develop portable applications that are compatible with the requirements and restrictions an application programmer must adhere to, in order to ensure compatibility with a JTWI-compliant device, including resource minimums (eg. standard-size application), clock resolution, and the use of preferred MIME names as applicable to CLDC 1.0/1.1, MIDP 2.0, WMA 1.1, and MMAPI 1.1).

[JTWI] 2.1.6 - > Resource minimums

Screen size returned by `Canvas.getWidth()` and `Canvas.getHeight()` -> 125 X 125 pixels

Color depth returned by `Display.numColors()` -> 4096 colors (12 bits)

Pixel aspect ratio -> 1:1

Volatile memory for Java Runtime -> 256 KB

[JTWI] 2.1.7 - > Standard-size application

JAR Size (MIDlet suite JAR) -> 64 KB

Application Descriptor Size(JAD) -> 5 KB

RMS Data Size(Set by MIDlet-Data-Size attribute) -> 30 KB

If the storage requirements (JAR and JAD) cannot be met, the installation MUST failed and the JAR shouldn't be downloaded. The RMS Data Size must be reserved for the application.

[JTWI] 3.3 - > Clock resolution

The method `java.lang.System.currentTimeMillis()` must record the elapsed time in increments not to exceed 40 milliseconds. At least 80% of test attempts MUST meet the time elapsed requirement.

[JTWI] 3.5 - > Preferred MIME names

Must support at least the preferred MIME name as defined by JANA. If no preferred name has been defined, then the registered name must be used.

Section 2: CLDC (1.0 / 1.1)

Good articles:

K Virtual Machine APIs- Which APIs Come from the J2SE Platform- -
<http://developers.sun.com/techtopics/mobility/midp/articles/api/>

2.1 - Identify correct and incorrect statements or examples about the requirements and scope of the CLDC specification, including the differences between 1.0 and 1.1.

[CLDC] 2.2 - > Requirements

Hardware Requirements

- Defines only minimum limits.
- At least 160 Kilobytes of *non-volatile memory* for VM and CLDC libraries.
- At least 32 Kilobytes of *volatile memory* for VM (example: java heap).

Software Requirements

- CLDC specs assumes that a minimum host OS or kernel is available to manage the hardware.
- This host OS must provide at least one schedulable entity to run the VM.

J2ME Requirements

- As a configuration, CLDC shall only define a minimum complement or "the lowest common denominator" of Java technology.
- Additional features should be defined in a J2ME profile.
- Shall generally define a subset of J2SE. Shall only define the variances and differences compared to the full JLS and JVMMS.

[CLDC] 2.3 - > Scope

Shall address:

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

Shall NOT address:

- Application installation and lyfe-cycle management
- User interface
- Event handling
- High-level application model.

[CLDC] 1.2 - > Differences between 1.0 and 1.1

- Floating point supported.
- Weak references supported.
- Classes Calendar, Date and TimeZone have been redesigned.
- New error class: NoClassDefFoundError.
- Thread object have names (Thread.getName()). Thread class has a few new constructors.
- Have been included:
 - . Boolean.TRUE and Boolean.FALSE
 - . Date.toString()
 - . Random.nextInt(int n)
 - . String.intern()
 - . String.equalsIgnoreCase()
 - . Thread.interrupt()
- Minimum memory budget increased from 160 Kb to 192 Kb.

2.2 - Describe the ways in which a CLDC virtual machine does and does not adhere to the Java Language Specification (JLS) and the Java Virtual Machine specification.

[CLDC] 4 - > JLS

- Do not include the method Object.finalize().
- Asynchronous exceptions are not supported.
- Set of error classes included in CLDC is limited because of two reasons:
 - . recovery from error conditions is usually highly device-specific.
 - . class java.lang.Error and its subclasses are exception from which programs are not ordinarily expected to recover.
- CLDC shall support the set of Errors classes derived from J2SE. When encountering any other error, the implementation shall behave as follows:
 - . either the VM halts in an implementation-specific manner
 - . or the VM throws the error nearest to CLDC error superclass.

If the CLDC VM implementing additional error classes that are part of the full JLS but

aren't required by CLDC spec, the implementation shall throw the error nearest CLDC supported superclass error.

[CLDC] 5 - > JVM5

Features eliminated

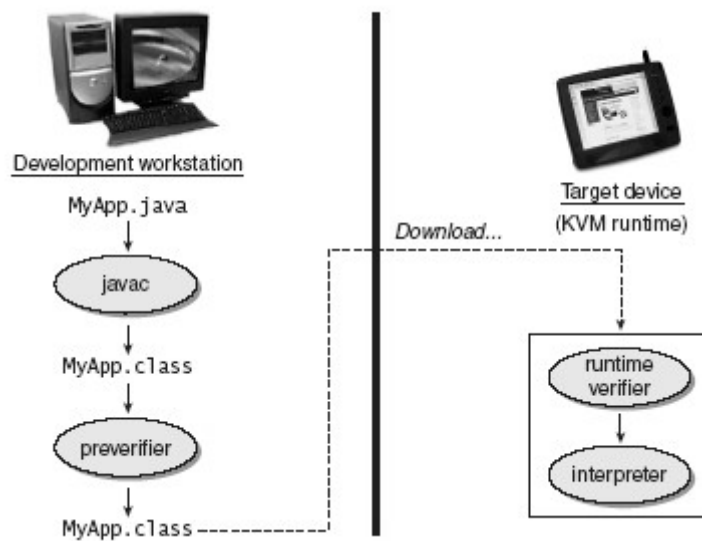
- User-defined class loaders - the device must have a built-in "bootstrap" class loader that cannot be overridden, replaced or reconfigured.
- Thread groups and daemon threads - thread operations can be applied to individual threads.
- Finalization of class instances
- Asynchronous exceptions - do not include the method `Object.finalize()`

Class file verification

- must able to reject invalid class files.
- can be implemented in two ways:
 - [1] - Using the standard class file verification defined in JVM5.
 - [2] - Using the alternative, more efficient, two phase pre-verification.

Off-device preverification and runtime verification with stack maps

Two phases:



First class files runs through a preverifier tool to remove certain bytecodes and add StackMap attributes. Each StackMap attributes consists of multiple entries, with each entry recording the types of local variables and operand stack items at a given bytecode offset.

At runtime the runtime verifier component of the virtual machine uses the additional StackMap attributes to perform the class verification efficiently.

Note that the preverified classes are fully compatible with the Java larger environments (J2SE and J2EE). The StackMap attribute is ignored by J2SE verifier.

All the subroutines in the bytecodes of class files are inlined.

Class File Format and Class Loading

- CLDC implementation MUST support compressed Java Archive (JAR) files.
- The jar files must contain preverified class files. Additionally, the JAR file may

contain application-specific resources files that can be loaded into the virtual machine by calling method `Class.getResourceAsStream(String name)`.

- About class file lookup order: The CLDC specification assumes class file lookup order is implementation-dependent. A virtual machine conforming to CLDC is not required to support the notion of classpath, but may do so at the implementation level.

- Two restriction about class file lookup order:

[1] - CLDC must guarantee that the application programmer cannot override, modify, or add system classes.

[2] - the programmer must not able to manipulate the class file lookup order in any way.

2.3 - Identify correct and incorrect statements or examples about CLDC classes including those derived from J2SE, and the CLDC-specific classes, including identifying which core J2SE classes are NOT included in CLDC, or have different behaviors (example: `java.lang.String`, io classes, etc.)

[CLDC] 6.2 - > J2SE

java.lang

Classes present:

Object, Class, Runtime, System, Thread, Runnable(interface), String, StringBuffer, Throwable, Boolean, Byte, Short, Integer, Long, Float, Double, Character, Math, ref.Reference, ref.WeakReference

Errors/Exception present:

Exception, ArithmeticException, ArrayIndexOutOfBoundsException, ArrayStoreException, ClassCastException, ClassNotFoundException, IllegalAccessException, IllegalArgumentException, IllegalMonitorStateException, IllegalThreadStateException, IndexOutOfBoundsException, InstantiationException, InterruptedException, NegativeArraySizeException, NullPointerException, NumberFormatException, RuntimeException, SecurityException, StringIndexOutOfBoundsException, Error, NoClassDefFoundError, OutOfMemoryError, VirtualMachineError.

Notes:

- No User Classloading is permitted.
- No Object Finalization.
- No Reflection API
- No Native Methods
- Multithreading - no groups or daemons.
- String and StringBuffer - don't have `compare(Object)` neither `compareToIgnoreCase(String)`
- Math - static methods, CLDC 1.0 don't has Float end Double operations.

Runtime and System

- greatly reduced from their J2SE counterparts.
- Calling `exit()` results in a `SecurityException`.
- Runtime does not support `exec()`.
- System don't have `System.in` (there is no console in a device). Has `System.out` and `System.in`
- `System.getProperty(String key)` return system properties. The key can be:

- > `microedition.platform`: contains the name of the device or host platform. If not supported returns null.
- > `microedition.encoding`: contains the default character encoding.
- > `microedition.configuration` -> contains the name of the implemented J2ME configuration.
- > `microedition.profiles` -> contains the names of the implemented J2ME profiles.

java.io

Classes present:

`InputStream`, `OutputStream`, `ByteArrayInputStream`, `ByteArrayOutputStream`, `DataInput`(interface), `DataOutput`(interface), `DataInputStream`, `DataOutputStream`, `Reader`, `Writer`, `InputStreamReader`, `OutputStreamWriter`, `PrintStream`

Error/Exceptions present:

`EOFException`, `InterruptedException`, `IOException`, `UnsupportedEncodingException`, `UTFDataFormatException`

Notes:

- There isn't the concept of local file system. So any class having to do with files have been pruned from `java.io` (`File`, `FileInputStream`, ...)
- Object Serialization is not supported. `Serializable` interface isn't present.
- `InputStreamReader` and `OutputStreamWriter` handle the conversion between byte streams and characters streams. An encoding on the constructor is optional, if no supplied the default is used.

java.util

Classes present:

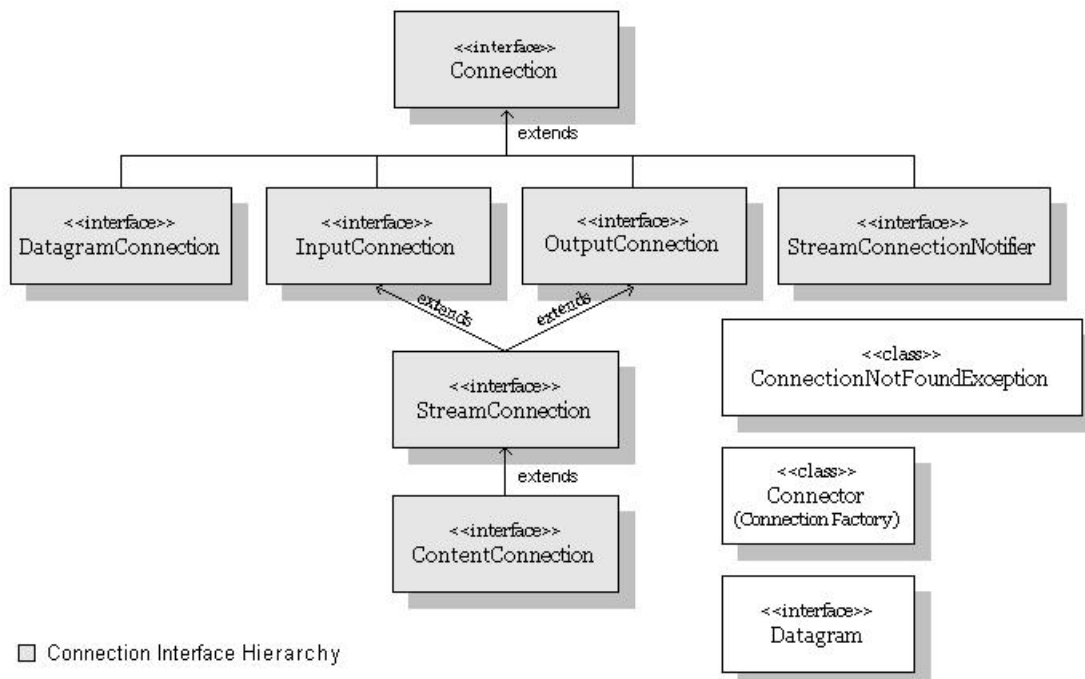
`Vector`, `Stack`, `Hashtable`, `Enumeration` (interface), `Calendar`, `Date`, `Time`, `Random`

Error/Exceptions present:

`EmptyStackException`, `NoSuchElementException`

[CLDC] 6.3 - > CLDC specific

Basically is all about the Generic Connection Framework:



2.4 - Given the differences and limitations of exception/error handling with CLDC devices, handle exceptions correctly.

[CLDC] 4.2 - > **Exception / Error** See objective 2.2 above.

2.5 - Write code that effectively manages memory / garbage collection.

Runtime.gc() - tell JVM to run the garbage collector. Isn't a static method.

System.gc() - shortcut to Runtime.gc(). Is a static method.

* I think that will be necessary some real code to get this objective very well.

Section 3: Security (both CLDC and MIDP)

Good articles:

Understanding MIDP 2.0's Security Architecture -

<http://developers.sun.com/techtoc/mobility/midp/articles/permissions/>

MIDP Application Security 3- Authentication in MIDP -

<http://developers.sun.com/techtoc/mobility/midp/articles/security3/>

3.1 - Given a set of requirements, design and build applications given CLDC-specified application-level security, including the sandbox model.

[CLDC] 3.4.2

By application-level security, we mean that a Java application can access only those libraries, system resources and other components that the device and the Java application environment allows it to access.

Sandbox model

- An application must run in a closed environment in which the application can access only those libraries that have been defined by the configuration, profiles and other classes supported by the device.
- Ensures that a malicious code cannot gain access to system resources.
- Sandbox model requires:
 - . Class files must be verified and guaranteed to be valid Java applications.
 - . The application programmer CANNOT modify or bypass the standard class loading mechanisms of the VM.
 - . A closed, predefined set of Java APIs is available to the application programmer as defined by CLDC, profiles and manufacturer-specific classes.
 - . The set of native functions is closed. The programmer cannot download any new libraries or access any native functions that are not part of the Java libraries provided by CLDC, profiles and manufacturer-specific classes.

Protecting system classes

The application programmer cannot override, modify, or add any classes to the packages java.*, javax.microedition or other profile-specific or manufacturer-specific packages.

Additional restrictions on dynamic class loading

The class loading mechanism of a VM conforming to CLDC is implementation-independent, with one important restriction: a Java application can load applications

classes ONLY from its own JAR. This ensures that Java applications on a device cannot interfere with each other or steal data from each other.

3.2 - Identify correct and incorrect statements or examples about untrusted MIDlet suites.

[MIDP] 3

Definition: Untrusted MIDlet suite is a MIDlet suite for which the origin and the integrity of the JAR file CANNOT be trusted by the device.

The untrusted domain MUST allow, WITHOUT explicit confirmation by user, access to:

API	Description
javax.microedition.rms	RMS APIs
javax.microedition.midlet	MIDlet Lifecycle APIs
javax.microedition.lcdui	User Interface APIs
javax.microedition.lcdui.game	The Game APIs
javax.microedition.media javax.microedition.media.control	The multi-media APIs for playback of sound

The untrusted domain MUST allow, WITH explicit confirmation by user, access to:

API	Protocol
javax.microedition.io.HttpConnection	http
javax.microedition.io.HttpsConnection	https

3.3 - Explain trusted MIDlet suite security authorization and permissions, including the process for MIDlet suite signing.

[MIDP] 3

I think the chapter 3 , page 41 to 43 of the Knudsen's book (2nd ed) is very important to understand this objective. The spec is too vague in this feature... Don't miss the articles with practical examples.

The MIDP 2.0 provides a new method:

```
public final int checkPermission(String permission)
```

This method returns 1 if the permission is granted and 0 if the permission is denied. A value return of -1 indicates that the implementation cannot determine if the permission will be granted or denied. Maybe the user will be asked about the given permission.

Protection Domain

MIDlets suites belong to protection domains that determine which permission are granted, which are denied, and which ones must be deferred to the user's judgment. The internal representation of protection domains and permissions is implementation

specific. A protection domain consists of

- . A set of permissions that should be allowed (Allowed). Do not require any user interaction.
- . A set of permissions that the user may authorize(User); each with its user interaction mode. Prompt user.

User Permission Interaction Mode

- blanket: is valid for every invocation of an API by a MIDlet suite until it is uninstalled or permission is changed by the user.
- session: is valid from the invocation of a MIDlet suite until terminates. MUST prompt the user on or before the first invocation of the API.
- oneshot: MUST prompt the user on each invocation of the API.

Requesting Permissions for a MIDlet

If the permission are critical to the function of the MIDlet suite and it will not operate correctly without them, use the attribute MIDlet-Permissions: permission_name on the descriptor.

If the MIDlet can work with limited capacity even without the permission, use the attribute MIDlet-Permissions-Opt: permission_name
Multiple permissions are separated by a comma.

Granting Permission to Trusted MIDlet Suites

All of the following MUST be true:

- The MIDlet suite must have been bound to a protection domain.
- The requested critical permissions are retrieved from the attributes MIDlet-Permissions and noncritical permissions from MIDlet-Permissions-Opt. If these attributes appear in the application descriptor they MUST be identical to corresponding attributes in the manifest. If they are not identical, the MIDlet suite MUST NOT be installed or invoked.
- If any of the requested permissions are unknown to attribute there are two possibilities. If they are not critical then they are removed from the requested permissions else (they are critical) the MIDlet suite MUST NOT be installed or invoked.
- If any of the requested permissions are not present in the protection domain (Allowed or User) there are two possibilities: If they are marked as critical the application MUST NOT be installed or invoked. Otherwise the application MUST still be installed and MUST be able to be invoked by user.
- If any requested permissions match the User permission of the protection domain then the user MUST explicitly provide authorization to grant those permissions to the MIDlet suite.
- The permissions granted to the MIDlet suite are the intersection of the requested permissions with the allowed and user granted permissions.
- During execution, any protected APIs MUST check for the appropriate permissions and throw a SecurityException if the permission has not been granted.

3.4 - Explain requirements and process of using X.509 PKI authentication for MIDlet suites.

[MIDP] 4

Requirements

- MUST support X.509 certificates and corresponding algorithms.
- MAY support additional signing mechanisms and certificate formats.
- When an attribute appears in the manifest (inside the JAR) it MUST NOT be overridden by a different value from the application descriptor.
- For trusted MIDlet suites the value in the application descriptor must equal to the value of the corresponding attribute in the manifest. If not, the MIDlet suite MUST NOT be installed.
- The MIDlet.getAppProperty method MUST return the attribute value from the manifest if one is defined. If not, the value from the application descriptor (if any) is returned.

Process

[1] - Creating the Signing Certificate

- The signer will need to contact the appropriate certificate authority.
- Example: the signer sends its distinguished name (DN) and public key to a certificate authority.
- The CA creates a RSA X.509 certificate and returns it to the signer.
- If multiple CA's are used then all the signer certificates in the application descriptor MUST contain the same public key.

[2] - Insert Certificates into the application descriptor

- The root certificate will be found on the device.
- Each certificate in the path is encoded (base64) and inserted into the application descriptor:

MIDlet-Certificate-<n>-<m> 'n' and 'm' defines the sequence path in which the certificates are tested.

[3] - Creating the RSA SHA-1 signature of the JAR

- The signature of the JAR is created with the signer's private key.
- The signature is base64 encoded and it's inserted in the application descriptor (JAD) :
MIDlet-Jar-RSA-SHA1:<base64 encoded signature>

[4] - Authenticating a MIDlet Suite

- When a MIDlet suite is downloaded the device MUST check if authentication is required.
- If the attribute MIDlet-Jar-RSA-SHA1 is present in the JAD then the JAR (or the suite) MUST BE authenticated verifying the signer certificates and JAR signature as below. Else (MIDlet-Jar-SHA1 isn't present) the application is installed but invoked as untrusted MIDlet suite.

[5] - Verify Signer

n starts with the value 1.

5.1 - Get the certification path from the application attribute MIDlet-Certificate-n-m where m starts from 1 and is incremented by 1 until there is no attribute with the given value.

5.2 - Validate the certificate path using protection domains as the authoritative source of protection domain root certificates. Bind the MIDlet suite to the protection domain that contains the protection domain root certificate that validates the first chain from signer to root and PROCEED with installation.

5.3 - If no full certification path could not be established after verifying MIDlet Certificate-<n>-<m>, increment n and repeat the steps 5.1 and 5.2.

[6] - Verify the JAR suite

- Gets the public key from the certificate obtained from the step 5.
- Gets the MIDlet-Jar-RSA-SHA1 attribute from the application descriptor.
- Decode the attribute from base64 and obtain a PKCS signature.
- Use the signers public key, the signature and SHA-1 digest of the JAR to verify the signature. If fails MUST NOT install the JAR or allow MIDlet's from the suite be invoked.

Summary of the verifications results:

Initial state	Verification result
JAD not present, JAR downloaded	Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted
JAD present but is JAR is unsigned	Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted
JAR signed but no root certificate present in the keystore to validate the certificate chain	Authentication can not be performed, JAR installation is not allowed
JAR signed, a certificate on the path is expired	Authentication can not be completed, JAR installation is not allowed
JAR signed, a certificate rejected for reasons other than expiration	JAD rejected, JAR installation is not allowed
JAR signed, certificate path validated but signature verification fails	JAD rejected, JAR installation is not allowed
JAR signed, certificate path validated, signature verified	JAR installation is allowed

Section 4: Networking

Good Articles:

[One of CLDC's innovations is the Generic Connection Framework - http://developers.sun.com/techttopics/mobility/midp/articles/genericframework/](http://developers.sun.com/techttopics/mobility/midp/articles/genericframework/)

[J2ME Low-Level Network Programming with MIDP 2.0 - http://developers.sun.com/techttopics/mobility/midp/articles/midp2network/](http://developers.sun.com/techttopics/mobility/midp/articles/midp2network/)

[Using Threads in J2ME Applications - http://developers.sun.com/techttopics/mobility/midp/articles/threading2/](http://developers.sun.com/techttopics/mobility/midp/articles/threading2/)

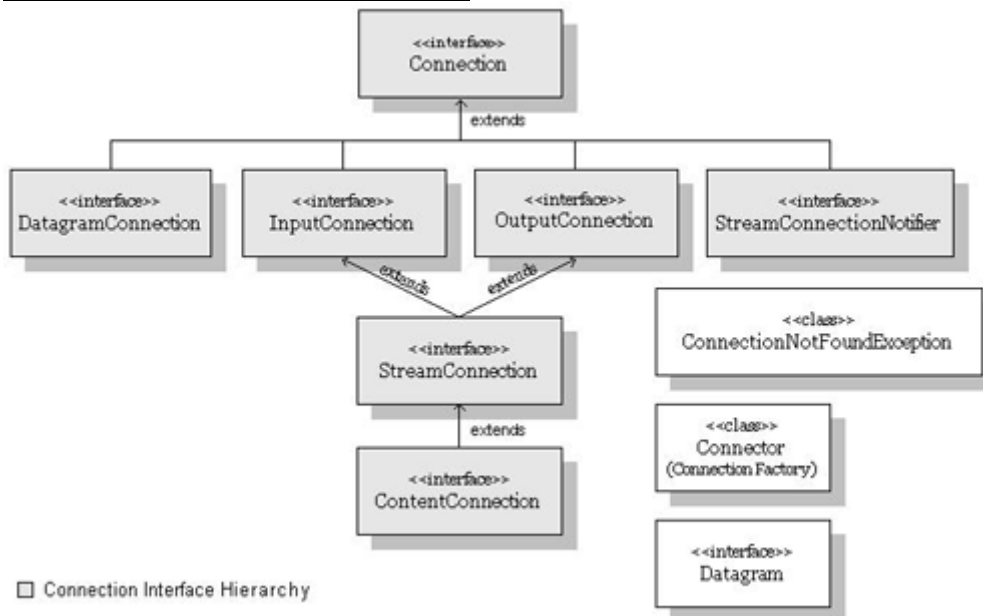
[Networking, User Experience, and Threads - http://developers.sun.com/techttopics/mobility/midp/articles/threading/](http://developers.sun.com/techttopics/mobility/midp/articles/threading/)

[MIDP Network Programming using HTTP and the Connection Framework - http://developers.sun.com/techttopics/mobility/midp/articles/network/](http://developers.sun.com/techttopics/mobility/midp/articles/network/)

4.1 - Write code using the Generic Connection framework specified by CLDC, recognizing its characteristics, use, classes, and interfaces. This may include identification of the class hierarchy and relationships of the Generic Connection framework.

[CLDC] 6.3.2

Class hierarchy and relation ships



Interface's Descriptions

Connection: most basic connection type that can only be opened and closed. The open method is not included in the interface (see Connector below)

InputConnection: basic serial input connection. Represents a device from which data can be read.

OutputConnection: basic serial output connection. Represents a device from which data can be written.

StreamConnection: circuit oriented connection. combines the InputConnection and OutputConnection interfaces. Forms a logical starting point for classes that implement two-way communication interfaces.

ContentConnection: provides access to some of the most basic metadata information provided by HTTP connections.

StreamConnectionNotifier: used when waiting for a connection to be established. The method acceptAndOpen will block until a client makes a connection. The method acceptAndOpen returns a StreamConnection.

DatagramConnection: datagram oriented connection. The address used for opening the connection is the endpoint at which datagrams are received. The destination for datagrams to be sent is placed in the datagram object itself.

Exception ConnectionNotFoundException

This exception is thrown when the device don't implement the protocol specified.

The Connector

- It's the connection factory.
- The Connection's interfaces don't have an open method. This task is responsibility of the Connector interface, by the static method:

```
Connector.open("<protocol>:<address>;<parameters>");
```

If successful, this method will return an object that implements one of the Connection's interface hierarchy.

Determining the protocol and implemented interface returned

- Determined by the string <protocol> of the method above.

socket://<hosts>:<port>;<parameters> -> javax.microedition.io.SocketConnection

socket://:<port>;<parameters> -> javax.microedition.io.ServerSocketConnection //if no port the system automatica allocates a available port

http://... -> javax.microedition.io.HttpConnection

https://... -> javax.microedition.io.HttpsConnection

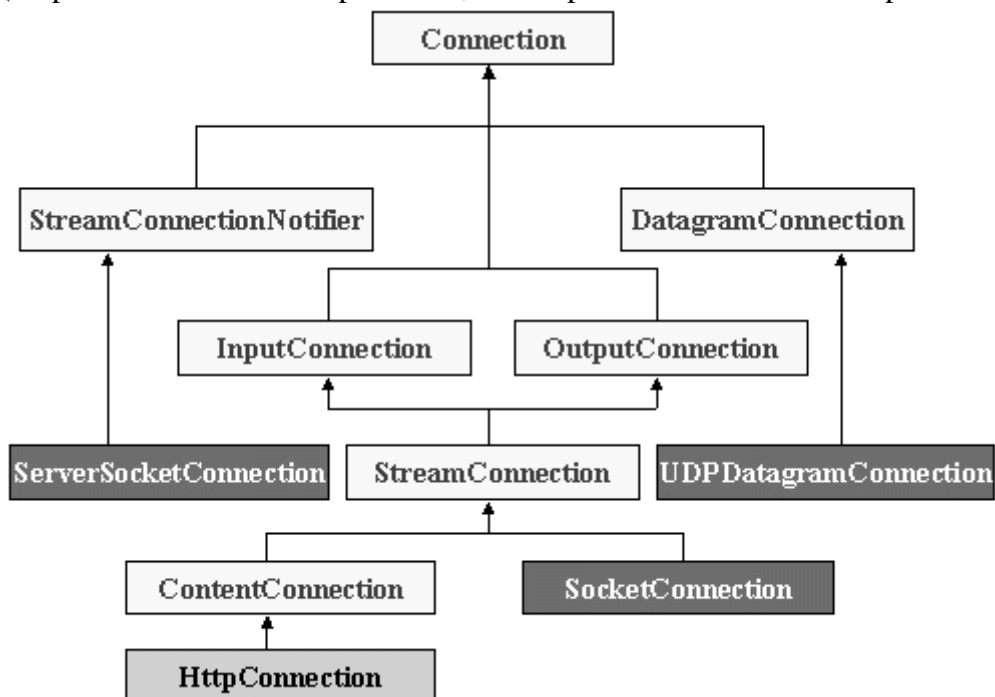
datagram://... -> javax.microedition.io.UDPDatagramConnection //if no port the system automatica allocates a available port

ssl://... -> javax.microedition.io.SecureConnection

comm://... -> javax.microedition.io.CommConnection

See in the next image how this is positioned in Generic Connection Framework

(HttpsConnection is not represented, but HttpConnection extends HttpConnection).



REMEMBER: ServerSocketConnection, UDPDatagramConnection, HTTPConnection, HTTPSConnection and SocketConnection was included in MIDP specification, not in CLDC specification.

The binding of protocols is done at runtime. At the implementation level, the string (up to the first occurrence of ':') that is provided as the parameters to the method Connector.open instructs the system to obtain the desired protocol implementation from a location where all the protocol implementations are stored.

Class/interfaces to read/write data from package java.io

InputStream, OutputStream, ByteArrayInputStream, ByteArrayOutputStream, DataInput(interface), DataOutput(interface), DataInputStream, DataOutputStream, Reader, Writer, InputStreamReader, OutputStreamWriter, PrintStream

4.2 - Write code for MIDP 2.0 networking, and issues and limitations related to HTTP, HTTPS, and TCP/IP sockets and Datagrams, recognizing which connections are required and which are optional, as well as comparing the issues related to TCP/IP and UDP Datagrams.

[MIDP] 7

The chapter 9 of Knudsen's Book is great to understand this objective too.

Required/Optional

GCF Interfaces, Classes, Exceptions	Vendors Must Support?	CLDC 1.0, 1.1	MIDP 1.0	MIDP 2.0
CommConnection	N			X
Connection (base connection)	Y	X	X	X
Content Connection	Y	X	X	X
Datagram Connection	N	X	X	X
HttpConnection	Y		X	X
HttpsConnection	Y			X
InputConnection	Y	X	X	X
OutputConnection	Y	X	X	X
SecureConnection	N			X
ServerSocket Connection	N			X
SocketConnection	N			X
StreamConnection	Y	X	X	X
StreamConnection Notifier	Y	X	X	X
UDPDatagram Connection	N			X

Concluding: MIDP 2.0 Requires only HTTP(HttpConnection) and HTTPS(HttpsConnection) plus the basic Generic Connection Framework interfaces excluding the DatagramConnection.

HTTP - interface HttpConnection

There are three states for an HTTP connection:

1 - Setup : where the request parameters can be set. The methods setRequestMethod and setRequestProperty can be invoked only in this state. To pass from the setup state to

connected state invoke any method that requires data to be sent or received by the server.

The following methods cause the state change in connected: `openInputStream`, `openDataInputStream`, `getLength`, `getType`, `getEncoding`, `getHeaderField`, `getResponseCode`, `getResponseMessage`, `getHeaderFieldInt`, `getHeaderFieldDate`, `getExpiration`, `getDate`, `getLastModified`, `getHeaderField`, `getHeaderFieldKey`. The following methods can be invoked when in setup or connected state: `close`, `getRequestMethod`, `getRequestProperty`, `getURL`, `getProtocol`, `getHost`, `getFile`, `getRef`, `getPort`, `getQuery`.

2 - Connected: request parameter have been sent and the response is expected. After an output stream has been opened by the `openOutputStream` or `openDataOutputStream` methods, attempts to change the request parameters via `setRequestMethod` or the `setRequestProperty` are ignored. Once the request parameters have been sent, these methods will throw an `IOException`. When an output stream is closed via the `OutputStream.close` or `DataOutputStream.close` methods, the connection enters the Connected state. When the output stream is flushed via the `OutputStream.flush` or `DataOutputStream.flush` methods, the request parameters **MUST** be sent along with any data written to the stream.

3 - Closed: final state, in which the HTTP connection as been terminated. The transition to Closed state from any other state is caused by the `close` method and the closing all of the streams that were opened from the connection.

HTTPS - interface `HttpsConnection` extends `HttpConnection`

Add two methods to `HttpConnection`: `int getPort()`, `SecurityInfo getSecurityInfo()`
In addition to the normal `IOExceptions` that may occur during invocation of the various methods that cause a transition to the Connected state, `CertificateException` (a subtype of `IOException`) may be thrown to indicate various failures related to establishing the secure link. The secure link is necessary in the Connected state so the headers can be sent securely. The secure link may be established as early as the invocation of `Connector.open()` and related methods for opening input and output streams and failure related to certificate exceptions may be reported.

SOCKETS

If the string have a Host then the interface implemented is **`SocketConnection`**

Closing streams: Every `StreamConnection` provides a `Connection` object as well as an `InputStream` and `OutputStream` to handle the I/O associated with the connection. Each of these interfaces has its own `close()` method. For systems that support duplex communication over the socket connection, closing of the input or output stream **SHOULD** shutdown just that side of the connection. e.g. closing the `InputStream` will permit the `OutputStream` to continue sending data. Once the input or output stream has been closed, it can only be reopened with a call to `Connector.open()`. The application will receive an `IOException` if an attempt is made to reopen the stream.

If the string don't have a host the interface is **`ServerSocketConnection`**. The port may be omitted (`sockets://`), in this case the system allocates the port (invoke `getPort()` to know the port). The method `acceptAndOpen()` returns a `SocketConnection` of the client.

DATAGRAMS - interface UDPDatagramConnection

- No guarantee that the packets will reach their destination in the right order, or that they will even arrive at all.
- All data is exchanged using Datagrams.
- To send a datagram, first use DatagramConnection to create one for you using one of the newDatagram() methods. Then write some data into it and pass it (the Datagram object) to the send() method of DatagramConnection.
- To receive call receive() . Blocks until a datagram is received.
- Datagram is an extension of DataInput and DataOutput interfaces. It is possible to read and write data as we do in a stream.

4.3 - Write code using the MIDP 2.0 classes in the javax.microedition.io package, including code that correctly opens, closes, and uses a network connection, using the implications of network blocking operations, scheme, connection number limitations, and character encoding.

[MIDP] 7

Open

Always use Connector.open("<protocol://host:port;parameters"). May throw ConnectionNotFoundException.

There is a way to get direct access to input/output streams. This way you will be restricted to the methods of the stream, because you don't have a reference to a Connection implementation. Use one of the following methods:

InputStream Connector.openInputStream(String url)

OutputStream Connector.openOutputStream(String url)

DataInputStream Connector.openDataInputStream(String url)

DataOutputStream Connector.openDataOutputStream(String url)

Close

Use the method inherited from the interface Connection, close().

Send and Receive Data

On streamed connections:

Use the classes InputStream, OutputStream, ByteArrayInputStream, ByteArrayOutputStream, DataInput(interface), DataOutput(interface), DataInputStream, DataOutputStream, Reader, Writer, InputStreamReader, OutputStreamWriter, PrintStream like in J2SE.

On datagram connections:

To send (Exceptions omitted):

```
DatagramConnection dc = Connector.open("datagram://x:1260"); // 1 - open connection
```

```
Datagram dat = dc.newDatagram(); //2 - creates the Datagram
```

```
...//3 - write on the datagram object
```

```
dc.send(dat); // 4 - sends
```

To receive (Exception omitted)

```
DatagramConnection dc = Connector.open("datagram://:1260"); // 1 - open connection
```

```
Datagram dat = null; //2 - reference to the Datagram
```

```
dc.receive(dat); // 3 - blocks until a datagram is received
```

4.4 - Given a problem scenario, troubleshoot networking issues for MIDP 2.0.

I think some real coding is required here. Don't forget the articles [Networking](#), [User Experience](#), and [Threads](#) and [Using Threads in J2ME Applications](#). They are great!

Section 5: Application Model /Delivery / Lifecycle / Provisioning

Good articles:

Understanding J2ME Application Models -

<http://developers.sun.com/techttopics/mobility/midp/articles/models/>

Deploying Wireless Java Applications -

<http://developers.sun.com/techttopics/mobility/midp/articles/deploy/>

5.1 - Explain the specification guarantees for: browsing for MIDlet suites, transferring MIDlet suites, using HTTP, push registries, basic authentication, installing and updating MIDlet suites, invoking MIDlet suites, and deleting MIDlet suites.

[MIDP] 2

Browsing for MIDlet suites

- The user SHOULD be able to access a network location and see a description of the MIDlet suite along with a link. If the link refers to a JAR the MAS starts the download and installation. If the link is for a Application Descriptor (JAD):

1 - the server MUST indicate in the response that the data has a MIME type of "text/vnd.sun.j2me.app-descriptor"

2 - then the AMS gets the application descriptor and validate it to verify if the installation would be possible. The AMS searches for required attributes and make conversion of encoding.

3 - the user SHOULD be given a chance to confirm the installation of the MIDlet suite. Any conditions that prevents the suite to be installed SHOULD be notified to the user.

Tranferring and installing the MIDlet suite

- During installation the user SHOULD be informed of the progress and MUST be able to cancel the installation, in this case the device MUST be left in the state it was in before installation began.

- If MIDlet is already installed , the installation SHOULD be treated as a update.

- The devices then initiates the download of the MIDlet suite via HTTP. If an JAD was first retrieved the URL must be exactly as defined in the JAD. Additional headers unnecessary.

- If the server or proxy responds with a 401 (Unauthorized) or 407 (Proxy Authentication Required) the user SHOULD provides credentials (like a username and password) and re-send the request.

- The MIDlet suite and the headers MUST be checked to verify that the retrieved MIDlet suite is valid and can be installed on the device. Many problems can occur and the user must be notified of some of them. See the objective 5.5 below.
- Provided that there are no problems, the MIDlets contained in the MIDlet suite MUST be installed and made available to the user.
- Installation is complete when the MIDlet was successfully installed or an unrecoverable failure has occurred. In either case, the status MUST be reported.

Updating MIDlet suites

- The user MUST be able to know the version of a suite. With this knowledge the user MUST be able to authorize or decline the installation.
- About the RMS RecordStore's associated with the MIDlet:
 - . MUST maintain the RecordStore if the cryptographic signer is the same for the new and old MIDlet suite or if the host, path and URL of the JAD or the suite is the same for both new and old.
 - . If false, then the user MUST be prompted if the data must be retained for the new MIDlet suite.
- An untrusted suite MUST NOT update a trusted suite.

Invoking MIDlet suites

- If the suite has more than one MIDlet, the user interface MUST allow the user to select each one for execution.

Deleting MIDlet suites

- Devices MUST allow the user removing suites.
- Special conditions during the removal MUST be notified to the user.
- If the attribute MIDlet-Delete-Confirm is present then the value must be included on the confirmation prompt the user.

5.2 - Identify correct and incorrect statements or examples about the MIDP application model, including: the MIDP execution environment, MIDlet suites, MIDlet suite packaging (including the manifest and the application descriptor), discovering available services on the device, discovering which version of MIDP/CLDC is on the device.

[MIDP] 12

MIDP execution environment

- shared by all MIDlets in a suite and any MIDlet can interact with other MIDlets packaged together. The AMS initiates the apps and makes the following available to the MIDlets:
 - .Classes and native code implementing CLDC, including a JVM.
 - .Classes and native code implementing the MIDP runtime.
 - .All classes from a single JAR file for execution.
 - .Non-class from a single JAR file as resources. (getResourceAsStream)
 - .Contents of the descriptor file, when its present. (getAppProperty).

- MIDlet suites MUST NOT contain classes that are in packages defined by the CLDC and MIDP. The application programmer CANNOT override, modify, or add any classes to these protected system packages.

MIDlet Suites

- The elements of a MIDlet suite are: runtime execution environment, MIDlet suite packaging, application descriptor and application lifecycle.

MIDlet suite packaging

- One or more MIDlets are packaged in a single JAR file that includes:
 - . A manifest describing the contents.
 - . Java classes for the MIDlets and classes shared by the MIDlets
 - . Resource files used by the MIDlets.

MIDlet Attributes

Attribute Name	Description
MIDlet-Name	Name of the suite
MIDlet-Version	Version number of the suite
MIDlet-Vendor	Organization that provides the suite
MIDlet-Icon	PNG filename
MIDlet-Description	Description of the suite
MIDlet-Info-URL	URL for further info about the suite
MIDlet-<n>	Name, icon, class of the MIDlet. Must be in sequence starting from 1. Broke the sequence terminates the list.
MIDlet-Jar-URL	URL from JAR file loading
MIDlet-Jar-Size	JAR size in bytes
MIDlet-Data-Size	Minimum storage required. Default is 0.
MicroEdition-Profile	Same of System property microedition.profiles.
MicroEdition-Configuration	Same of System property microedition.configuration
MIDlet-Permissions	Zero or more permissions that are critical
MIDlet-Permissions-Opt	Zero or more permissions that are not critical
MIDlet-Push-<n>	Register a MIDlet for Push
MIDlet-Install-Notify	
MIDlet-Delete-Notify	
MIDlet-Delete-Confirm	

Manifest MUST contain: MIDlet-Name, MIDlet-Version, MIDlet-Vendor

JAD MUST contain: The above plus MIDlet-Jar-URL, MIDlet-Jar-Size

Manifest or JAD MUST contain: MIDlet-<n>, MicroEditio-Profile,
MicroEdition-Configuration

The application developer MUST NOT add attributes that begin with MIDlet- or MicroEdition-

Duplication of Attributes

- MIDlet-Name, MIDlet-Version and MIDlet-Vendor MUST BE identical or the suite is NOT installed.
- If the suite is trusted the attributes duplicated MUST have the same value or else the suite is not installed.
- If the suite is untrusted the JAD will overwrite the manifest value.

Discovering Version

Format: Major.Minor[.Micro]. A missing MIDlet-Version tag is assumed to be 0.0.0

5.3 - Develop applications that correctly reflect a MIDlet's application lifecycle, including: the purpose of the MIDlet class, communication with the application management software, platform request API, valid MIDlet states and transitions, and the behavior that should and should NOT be implemented within different lifecycle methods (including the constructor).

[MIDP] 12

Purpose of the MIDlet class

extend by a MIDlet to allow the AMS to start. Stop, and destroy it.

Communication with AMS

request the arguments from the application descriptor.

Platform Request

Use the method:

```
public final boolean platformRequest(String URL)
    throws ConnectionNotFoundException
```

- This is a non-blocking method. If the platform has the resources it SHOULD bring the appropriate application to the foreground and let the MIDlet running in the background else the platform MAY wait to handle the request until the MIDlet suite exit. Then the platform MUST bring the appropriate application to handle the request. Only the last queue is

- Common platform requests are installation of MIDlet (the string target a JAR or JAD) like a MIDlet trying to update itself on the application and phone calls (by the string phone://tel).

Valid States of a MIDlet

Paused: SHOULD not be holding any shared resources. Entered by:

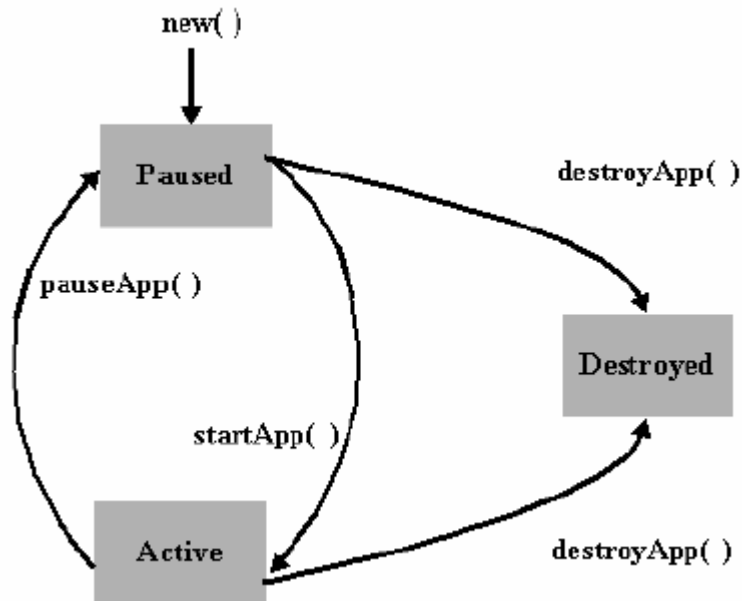
- After a MIDlet has been created using new.
- From the Active state after the MIDlet.pauseApp();
- From the Active state when the MIDlet.notifyPaused() returns.
- From the Active state if startApp throws a MIDletStateChangeException

Active: Functioning normally. Entered by:

- Just prior to the AMS calling the MIDlet.startApp();

Destroyed: Released all of its resources and terminated. Only entered once! Entered by:

- AMS called MIDlet.destroyApp() returns except in the case when the unconditional argument is false and a MIDletStateException is thrown.
- Call to MIDlet.notifyDestroyed(). This method DON'T call destroyApp() so the the MIDlet must have performed the equivalent of the MIDlet.destroyApp() method before calling MIDlet.notifyDestroyed().



MIDlet interface methods

pauseApp – the MIDlet must release any temporary resources and become passive

startApp – the MIDlet must acquire any resources

destroyApp – the MIDlet SHOULD save any state and release ALL resources

notifyDestroyed – the AMS is notified that the MIDlet has cleaned up and is done.

notifyPaused – the AMS is notified that the MIDlet is paused

resumeRequest – the MIDlet asks application AMS to be started again

getAppProperty – gets a named property from the MIDlet

Behavior that SHOULD'T be implemented

- If the MIDlet has a static void main (String args[]), must be ignored.
- If the MIDlet calls System.exit() a SecurityException will be thrown.

5.4 - Deploy a MIDP 2.0 application with the correct use of JAD files and manifests.

[MIDP] 12

The JAD file must have a .jad extension and its MIME type is “text/vnd.sun.j2me.app-descriptor”. See the attributes on objective 5.2 above.

Name is <MIDlet Suite Name>.jad

5.5 - Given an installation failure, analyze the problem, and develop possible resolutions.

Status Code	Description
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

5.6 - Given a set of requirements, develop applications that correctly implement MIDP 2.0 support for delayed or scheduled activities using timers and background threads.

Registering a Timer Alarm

Use the method:

```
public static long registerAlarm(String midlet, long timeInMili)
```

```
throws ClassNotFoundException, ConnectionNotFoundException
```

- Passing a time of zero will disable the alarm. Note that don't exists a method like removeAlarm or unregisterAlarm.

- Only one alarm for MIDlet is supported, and invoke this method twice overwrites the previous alarm .

- Note that the push registry will call the MIDlet only if the MIDlet isn't running. For timer tasks which the MIDlet is running use the classes Timer and TimerTask like the following code:

```
// cyclic background task info.
long REFRESH_TIME = 1000*60*5; // every 5 minutes
Timer aTimer = new Timer();
MyTask myTask = new MyTask();
aTimer.schedule(myTask, 0, REFRESH_TIME);
...
class MyTask extends TimerTask {
    // Constructor.
    public MyTask() {
    }
    ...
    // Thread run method.
    public void run() {
... Your Task Logic
    }
}
```

[MIDP] 6

Section 6: MIDP Persistent Storage

Good articles:

[MIDP Database Programming Using RMS- a Persistent Storage for MIDlets - http://developers.sun.com/techtopics/mobility/midp/articles/persist/](http://developers.sun.com/techtopics/mobility/midp/articles/persist/)

The chapter 8 of the Knudsen's book is very good to study this objective.

6.1 - Develop code that correctly implements handling, sharing and removing RecordStores within MIDlet suites.

[MIDP] 14

- RecordStores is implementation specific.
- When a MIDlet is removed, all RecordStores associated with the MIDlet MUST be removed too.
- RecordStores are identified by the descriptor attributes MIDlet-Name, MIDlet-Vendor and its name. It's possible to exist two RecordStores with the same name associated with different MIDlets. The name of a RecordStore consists of a combination of up to 32 Unicode characters.
- RecordStores are Thread-Safe (your operations are atomic, synchronous and serialized) however an operation can overwrite another in a multithreaded MIDlet situation. Be careful.
- The record store maintains a version number which can be retrieved by the method "int getVersion()". This number is updated every time the record store is modified.
- The record store also remembers the last time it was modified. Use the method "long getLastModified()". You can create a Date from the long and then you could use a Calendar with the Date.
- To list all RecordStores available to a suite use the static method "public String[] listRecordStores()"
- To get the total size of the store use "public int getSize()"
- To get the available storage space use "public int getSizeAvailable()"
- If the device doesn't have more space available then an operation can throw a RecordStoreFullException.
- To know the next record ID call getNextRecordID()
- To know how many records exist in the RecordStore call getNumRecords()

Handling & Sharing

- There are three methods to create/open a RecordStore.
- The boolean createIfNecessary determines if a new record store will be created in the case that it doesn't exist yet. If createIfNecessary is false and the RecordStore doesn't exist a RecordStoreNotFoundException will be thrown.
- The byte authMode can assume the constant values AUTHMODE_PRIVATE(default) and AUTHMODE_ANY(shared store). This argument is ignored if the RecordStore exists.
- The boolean writable determines if the MIDlets that have access granted can modify the store. This argument is ignored if the RecordStore exists.

- The attribute vendor corresponds to the descriptor attribute MIDlet-Vendor and the attribute suiteName corresponds to the descriptor attribute MIDlet-Name.
- If a MIDlet pertences to the creator's store suite then it can call the method "public void setMode(byte authMode, boolean writable) " to change the the authorization mode and writable flag.

```
static RecordStore openRecordStore(String recordStoreName, boolean
createIfNecessary)
```

```
static RecordStore openRecordStore(String recordStoreName, boolean
createIfNecessary, int authmode, boolean writable)
```

```
static RecordStore openRecordStore(String recordStoreName, String vendor, String
suiteName)
```

- The last opens a shared store from other suite, can throw RecordStoreNotFoundException. A MIDlet from a suite CANNOT delete a RecordStore from another suite.

Closing

Method:

```
public void closeRecordStore()throws RecordStoreNotOpenException,
RecordStoreException
```

- If the RecordStore isn't open, a RecordStoreNotOpenException will be thrown.
- When the record store is closed, all listeners are removed and all RecordEnumerations associated with it become invalid. If the MIDlet attempts to perform operations on the RecordStore object after it has been closed, the methods will throw a RecordStoreNotOpenException.

Removing

-Use the method:

```
public static void deleteRecordStore(String recordStoreName) throws
RecordStoreException, RecordStoreNotFoundException
```

- If this method is called when the RecordStore is open, then a RecordStoreException will be thrown. This method DON'T notifies any listener (DON'T NOTIFY the method recordDeleted of a RecordListener)

6.2 - Develop code that correctly implements adding, retrieving, modifying, and deleting individual records in a RecordStore, and converting RecordStore record data to and from byte arrays, and that reflects performance implications.

[MIDP] 14

RecordStore data to and from byte arrays

```
//from recordStore
byte[] data = recordStore.getRecord(1);
ByteArrayInputStream bi = new ByteArrayInputStream(data);
DataInputStream ip = new DataInputStream(bi);
int i = ip.readInt();
String str = ip.readUTF();

//to record store
int number = 10;
String aString = "Hello";
ByteArrayOutputStream bo = new ByteArrayOutputStream();
DataOutputStream do = new DataOutputStream(bo);
do.write(number);
do.writeUTF(aString);
byte[] data = bo.toByteArray();
recordStore.addRecord(data,0,data.length);
```

Adding

Use the method:

```
public int addRecord(byte[] data, int offset, int numBytes)throws
RecordStoreNotOpenException,RecordStoreException, RecordStoreFullException
```

- Return the record id recently added.

Retrieving

There are two method for retrieving data:

```
public byte[] getRecord(int recordId) throws
RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException
```

```
public int getRecord(int recordId, byte[] buffer, int offset)throws
RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException
```

Examples:

```
byte[] data = recordStore.getRecord(id);
or
byte[] data = new byte[recordStore.getRecordSize(id)];
recordStore.getRecord(id,data,0);
```

The second method is faster and more efficient.

You can create a byte array with the largest size of the records and use it as a buffer.

If the array you supply is not large enough to hold the record, and `ArrayOutOfBoundsException` will be thrown.

Modifying - overwrites data, if ID don't exists throws InvalidRecordIDException

Use the method of RecordStore:

```
public void setRecord(int recordId, byte[] newData, int offset, int numBytes) throws  
    RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException,  
    RecordStoreFullException
```

Deleting - Record ID is not reused, if ID don't exists throws InvalidRecordIDException

```
recordStore.deleteRecord(idRecord);
```

6.3 - Identify correct and incorrect statements or examples about filtering, comparing, event listening, and enumerating records in a RecordStore.

[MIDP] 14

Event Listening

The listener class must implement RecordListener.

To add or remove a listener to a RecordStore use the methods:

```
public void addRecordListener(javax.microedition.rms.RecordListener listener)  
public void removeRecordListener(javax.microedition.rms.RecordListener listener)
```

The methods of the RecordListener interface are self-explanatory:

```
void recordAdded(RecordStore recordStore, int recordId)  
void recordChanged(RecordStore recordStore, int recordId)  
void recordDeleted(RecordStore recordStore, int recordId)
```

Enumerating

- The interface is RecordEnumeration.
- It's created by the method of RecordStore, enumerateRecords(RecordFilter, RecordComparator, keepUpdate).
- If RecordFilter is null all the Record's are retrieved.
- If RecordComparator is null, the Record's won't be in any order.
- keepUpdate maintains the enumeration updated with the store. This is bad to performance, because the index are rebuilt every time that has a change to the store.
- Navigate through the enumeration using the methods nextRecord() and previousRecord(). If no more Records are available throws InvalidRecordIDException until reset() is called.
- To peek the id values use nextRecordId() and previousRecordId(). ATTENTION: This will deslocate "the pointer" of the Enumeration.
- To know the number of record use "int numRecords()"
- rebuild() reconstruct the RecordEnumeration.

Filtering

- Provides search capabilities to a RecordEnumeration
- The class must implement RecordFilter.
- Defines only one method :

boolean matches(byte[] candidate)

- This method must return true if the record must be included in the RecordEnumeration

Comparing

- Ordering the records in a RecordEnumeration.
- The class must implement RecordComparator
- Defines three attributes (that must be returned by the compare method) and one method:

static int EQUIVALENT

static int FOLLOWS

static int PRECEDES

int compare(byte[] rec1, byte[] rec2)

- If rec1 < rec2 PRECEDES must be returned
- If rec1 > rec2 FOLLOWS must be returned
- If rec1 = rec2 EQUIVALENT must be returned

Section 7: Push Registry

Good articles:

DON'T FORGET TO READ THIS ARTICLE. FOR ME, IS THE BEST MATERIAL ABOUT PUSH REGISTRY THAT I FOUND IT.

The MIDP 2.0 Push Registry -

<http://developers.sun.com/techtopics/mobility/midp/articles/pushreg/>

7.1 - Explain MIDP 2.0 Push Registry benefits, and limitations, and describe its use in applications.

[MIDP] 7

Benefits

- The push registry enables MIDlets to set themselves up to be launched automatically without user interaction.
- The push registry reacts to network events or time events.

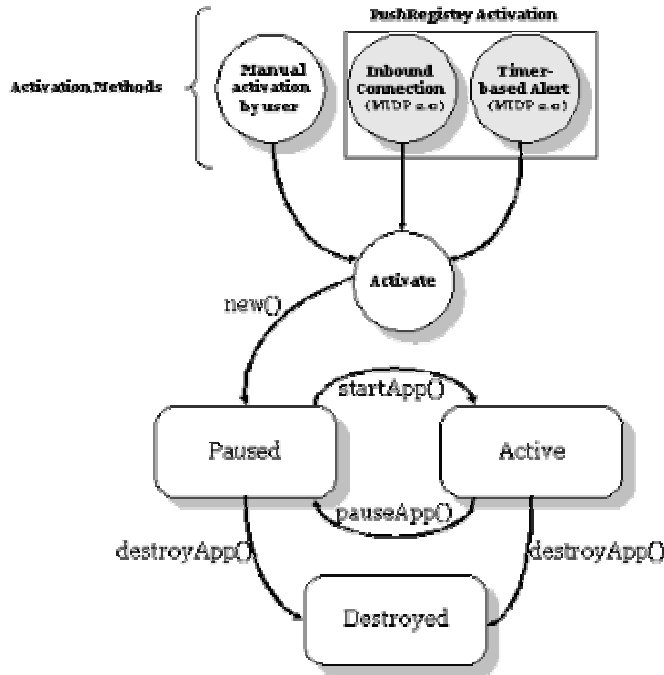
Limitations

- The application must be registered.
 - Only SMS do some buffering. For datagrams at least the first datagram must be buffered.
 - On connection streams based (like sockets) may happen timeouts.
 - Any error on descriptor registration causes the application NOT BE installed.
 - The system can support a protocol but not support the push registry protocol.
- Example: A device X support Connection for sockets but don't support sockets events in PushRegistry. In this case , at the time of registration, a ConnectionNotFoundException will be thrown.

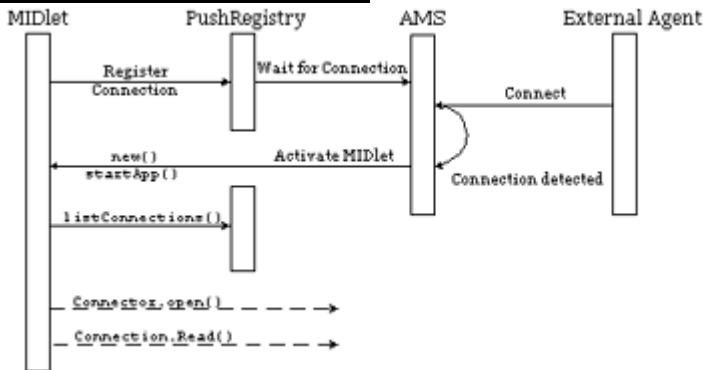
Using

- What will happen if a push event happens when another application is running is implementation specific. It MAY destroy or pause the application.
- The AMS only will listen for events when a application IS NOT running. The AMS WILL NOT listen even if the connection is closed, only will listen after the destroy.

Life Cycle



Network Activation Events



7.2 - Develop applications that correctly use MIDP 2.0 Push Registry including discovery, dynamic vs. static, and recognizing the types of connections that can and cannot be accepted.

[MIDP] 7

PushRegistry class

- Is part of the Generic Connection Framework.

Class definition: public class Pushregistry extends Object	
Method	Description
getFilter()	Returns the filter <code><filter></code> for the specified push connection
getMidlet()	Returns the Midlet responsible for handling the specified push connection
listConnections()	Returns the list of registered push connections for the Midlet suite

registerAlarm()	Registers a timer-based alarm to launch the Midlet. Disables alarms if an argument of zero is passed
registerConnection()	Registers a push connection
unregisterConnection()	Unregisters a push connection

Exceptions

Exception	Thrown by	Description
ClassNotFoundException	registerConnection() registerAlarm()	The specified MIDlet class name can't be found in the current MIDlet suite. In other words, the specified MIDlet class name was not defined (MIDlet-<n> attribute) in the descriptor file or the JAR file manifest, or the MIDlet argument is null.
ConnectionNotFoundException	registerConnection()	The platform does not support the specified connection type for push inbound connections.
	registerAlarm()	The platform does not support alarm-based application launch.
	Connector.open()	The requested protocol does not exist, or the connection could not be made.
IllegalArgumentException	registerConnection()	The connection string is not valid, or the filter string is not valid.
	Connector.open()	One of the arguments is invalid.
IOException	registerConnection()	The connection is already registered, or there are insufficient resources to handle the registration.
	Connector.open()	A generic I/O error was encountered.
SecurityException	registerConnection()	The specified MIDlet does not have permission to register a connection.
	unregisterConnection()	The specified connection was registered by another MIDlet suite.
	registerAlarm()	The specified MIDlet does not have permission to register an alarm.
	Connector.open()	The MIDlet has no permission to use the requested protocol.

Static Registration

- By attribute descriptors:
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
 - MIDlet-Push-<n>: n must be a sequence. A break in sequence stops the invocation in that point.
 - ConnectionURL: the connections on which the AMS will listen. String like the Connector's URL.
 - MIDletClassName: the fully qualified name of the MIDlet class.
 - AllowedSender: who can invoke the push registry to activate. May contain *(zero or more) and ? (one). Normally the value here is a ip.
 - Any error on the declaration of the attributes will result in NO INSTALLATION of the suite.
 - If the port is already request or reserved the application will NOT be installed.
 - The MIDletClassName MUST BE listed in one of the MIDlet-<n> attributes or the application will NOT be installed.
- REMEMBER: PushRegistry requires permission to use. The attributte MIDlet-Permission must be present or a SecurityException can be thrown by the application.

Dynamic Registration

Register a inbound connection

Open the Connection without a port. The system will allocate a port automatically.

After that discover the port with the method getPort() and register the Connection with:

```
static void registerConnection(String connection, String midlet, String filter) without a port.
```

Section 10: Media using MIDP 2.0 and the Mobile Media API 1.1 (MMAPI)

Good articles:

The J2ME Mobile Media API -

<http://developers.sun.com/techttopics/mobility/midp/articles/mmapioverview/>

MMAPI Overview -

http://developers.sun.com/techttopics/mobility/apis/articles/mmapi_overview/

10.1 - Given a set of requirements, develop code using MMAPI's support for tone generation.

- Tone generation is based in frequency and duration.
- There are two ways to do tone generation:
Manager.playTone(int note, int durationMilli, int volume) //note (0-127), durationMilli(), volume(1-100)
AND
You can also create a player for synthesizing tone sequences:
Player player = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR /* This have a value = device://tone */);
Then create a ToneControl for sinthesizing the tone.

Tone Sequence Format

- It's a byte array. Their values have meaning in pairs.
 - MUST start with ToneControl.VERSION, 1
 - The following are optional (INT = integer number):
 - . ToneControl.TEMPO, INT -> changes the tempo in bpm to INT*4 (example 15 = 60 bpm). Default is 120 bpm. Only at the start of sequence.
 - . ToneControl.RESOLUTION, INT -> changes the resolution to 1/INT (example 64 = 1/64 resolution). Default is 1/64 at the start of sequence.
 - . ToneControl.BLOCK_START, INT and ToneControl.BLOCK_END, INT -> Block definition. INT must be the same. Must be defined following VERSION, TEMPO and RESOLUTION.
 - . ToneControl.PLAY_BLOCK, INT -> plays the block defined by INT.
 - . ToneControl.SET_VOLUME, INT(1-100) -> sets the volume (percent). Any time in a sequence.
 - . ToneControl.SILENCE -> any time in a sequence.
 - . ToneControl.REPEAT, T, N, D -> repeat T times the note N with duration of D.
- To set a sequence to a control use ToneControl.setSequence(byte[] sequence). To start use the method Player.start().

10.2 - Given a set of requirements, develop code that correctly uses MIDP support for sound including audio playback, tone generation, media flow controls (start, stop, etc.), media type controls (volume, tone), and media capabilities using "Manager", "Player", and "Control" objects, recognizing the difference between required vs. optional features.

Required vs. optional features

Requirements set by the MIDP 2.0 expert group:

- Low footprint audio playback
- Protocol and content format agnostic.
- Support for tone generation.
- Support for general media flow controls: start, stop, seek, etc.
- Support for media-specific type controls: volume, etc
- Support for capability query.

Differs from general multimedia API in the following ways:

- It's audio only.
- It does not support simultaneous playback of multiples Player using a common time base.
- It does not support custom protocols at the application level. The javax.microedition.media.protocol is removed (DataSource).
- A simplified version of Manager is used.

The Player States

There are 5 states (there are static members variables int for them):

- UNREALIZED - Player implementation created, but hasn't tried to find the audio data and hasn't tried to acquire resources like the audio hardware.
- REALIZED - After it locates the media data.
- PREFETCHED - Get ready to start rendering audio.
- STARTED - Has begun rendering audio data.

- CLOSED - releases all resources and connections and cannot be used again.

States Methods

int Player.getState() - retorna o estado atual.

realize() - goes to REALIZED.

prefetch() - goes to PREFETCHED (implicit goes to REALIZED first, if necessary)

start() - goes to STARTED (implicit goes to PREFETCHED first, if necessary)

A call to a method is a implicit call to the early methods (call to start implies calls to realize and prefetch).

stop() - takes a STARTED player to PREFETCHED.

deallocate() - takes a PREFETCHED or STARTED player to REALIZED state. If a UNREALIZED player is stuck trying to locate media this method takes the PLAYER back to UNREALIZED.

close() - takes a player in any state to CLOSED.

Manager, Player and Control

- The Manager (acts as a factory) creates the Player. The Player gives access to controls.

Player static Manager.createPlayer(String url) // creates the player, can do network connection, MUST be permitted

Control[] Player.getControls(); //get availables controls

Control Player.getControl(String controlName); //get a control

Audio Playback

- Jump to a particular point in an audio clip: void Player.setMediaTime(long now);

- Know in which point is a media: long Player.getMediaTime();

- Know the total time of a media: long Player.getDuration();

- Loop a media (-1 loop infinite times): Player.setLoopCount(int times);;

Controls

To get a control use the methods (must be REALIZED at least):

Player Control[] getControls();

Player Control getControl(String controlName);

The ABB defines only VolumeControl and ToneControl controls.

ToneControl

This interface defines the byte constants used in tone sequences plus a method:

```
public void setSequence(byte[] sequence);
```

VolumeControl

Define 4 methods:

public int getLevel() -> get the actual level of the sound (1-100)

public boolean isMuted() -> self-explanatory

public void setMute(boolean mute) -> self-explanatory

public int setLevel(int level) -> set the level. If <0 the level will be 0. If >100 the level will be 100.

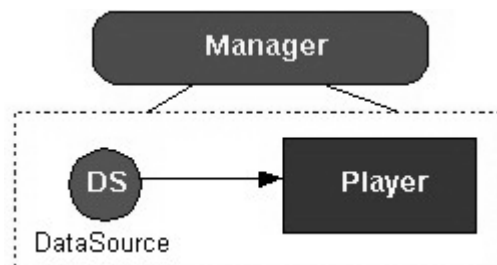
10.3 - Develop code that correctly uses MMAPI support for playback and recording of media, including the use of the "DataSource", "Player", and "Manager" objects, support for audio and video capture and playback, system properties queries, recognizing the difference between required and optional features.

Datasource, Player and Manager

A Datasource hides the detail of how the data is read from its source. The Datasource provides a set of methods to allow a Player to read data from it for processing.

A Player reads from the Datasource, processes the data, and renders the media to the output device. Provides a set of methods to control media playback and basic synchronization. Provide some type-specific controls.

A Manager is the factory mechanism, creates Players from Datasources. Manager also provides methods to create Players from locators and InputStreams.



Capture

String locator: "capture://video". Then use a method VideoControl.getSnapshot(String imageType).

The 'video.snapshot.encodings' key can be used to query the supported snapshot formats from the system.

System properties query

Obtained by the method System.getProperty(String key);

microedition.media.version -> 1.1

supports.mixing -> (true | false) If true supports at least two tones by Manager.playTone or Manager.playTone plus a Player or two Players.

supports.audio.capture -> (true | false) If true audio.encoding must not be null or empty.

supports.video.capture -> (true | false) If true video.encoding must not be null or empty.

supports.recording -> (true | false) If true at least one Player type supports recording.

audio.encodings -> Supported capture audio formats, delimited by at least one space. If audio capture is not supported MUST return null.

video.encodings -> Supported capture video formats, delimited by at least one space. If video capture is not supported MUST return null.

video.snapshot.encodings -> Supported video snapshots formats, delimited by spaces. If snapshot is not supported MUST return null.

streamable.contents -> Supported streamable content types.MIME syntax, each delimited by at least one space.

Required versus optional features

- Not all implementations of MMAPI support all multimedia types and input protocols.
- Manager provides `getSupportedContentTypes` and `getSupportedProtocols`.
- Does not mandate any particular media types or protocols

- Optional features are organized as Controls:

Feature Set	Implementation Requirements
Sampled Audio	SHOULD : VolumeControl, StopTimeControl
MIDI	SHOULD: VolumeControl, MIDIControl, TempoControl, PitchControl, StopTimeControl
Tone Sequence	MUST: ToneControl SHOULD: VolumeControl, StopTimeControl
Interactive MIDI	MUST: MidiControl
Video	MUST: VideoControl SHOULD: FramePositioningControl, StopTimeCControl, VolumeControl

10.4 - Identify correct and incorrect statements or examples about the media class hierarchies in both MIDP 2.0 and MMAPI 1.1.

MMAPI

Package Hierarchies:

javax.microedition.media, javax.microedition.media.control,

javax.microedition.media.protocol

Class Hierarchy

class java.lang.Object

class javax.microedition.media.protocol.ContentDescriptor

class javax.microedition.media.protocol.DataSource (impl Controllable)

class javax.microedition.media.Manager

class java.lang.Throwable

class java.lang.Exception

class javax.microedition.media.MediaException

Interface Hierarchy

interface javax.microedition.media.Control

interface javax.microedition.media.control.FramePositioningControl

interface javax.microedition.media.control.GUIControl

interface javax.microedition.media.control.VideoControl

interface javax.microedition.media.control.MetaDataControl

interface javax.microedition.media.control.MIDIControl

interface javax.microedition.media.control.PitchControl

interface javax.microedition.media.control.RateControl

interface javax.microedition.media.control.TempoControl

interface javax.microedition.media.control.RecordControl

interface javax.microedition.media.control.StopTimeControl

interface javax.microedition.media.control.ToneControl

interface javax.microedition.media.control.VolumeControl

interface javax.microedition.media.Controllable

interface javax.microedition.media.Player

interface javax.microedition.media.protocol.SourceStream

interface javax.microedition.media.PlayerListener

interface javax.microedition.media.TimeBase

MIDP 2.0

Class Hierarchy

class java.lang.Object

class javax.microedition.media.Manager

class java.lang.Throwable

class java.lang.Exception

class javax.microedition.media.MediaException

Interface Hierarchy

interface javax.microedition.media.Control

interface javax.microedition.media.control.ToneControl

interface javax.microedition.media.control.VolumeControl

interface javax.microedition.media.Controllable

interface javax.microedition.media.Player

interface javax.microedition.media.PlayerListener

Section 11: Wireless Messaging API 1.1 (WMA)

Good Articles:

[The Wireless Messaging API -](http://developers.sun.com/techtoc/mobility/midp/articles/wma/)

<http://developers.sun.com/techtoc/mobility/midp/articles/wma/>

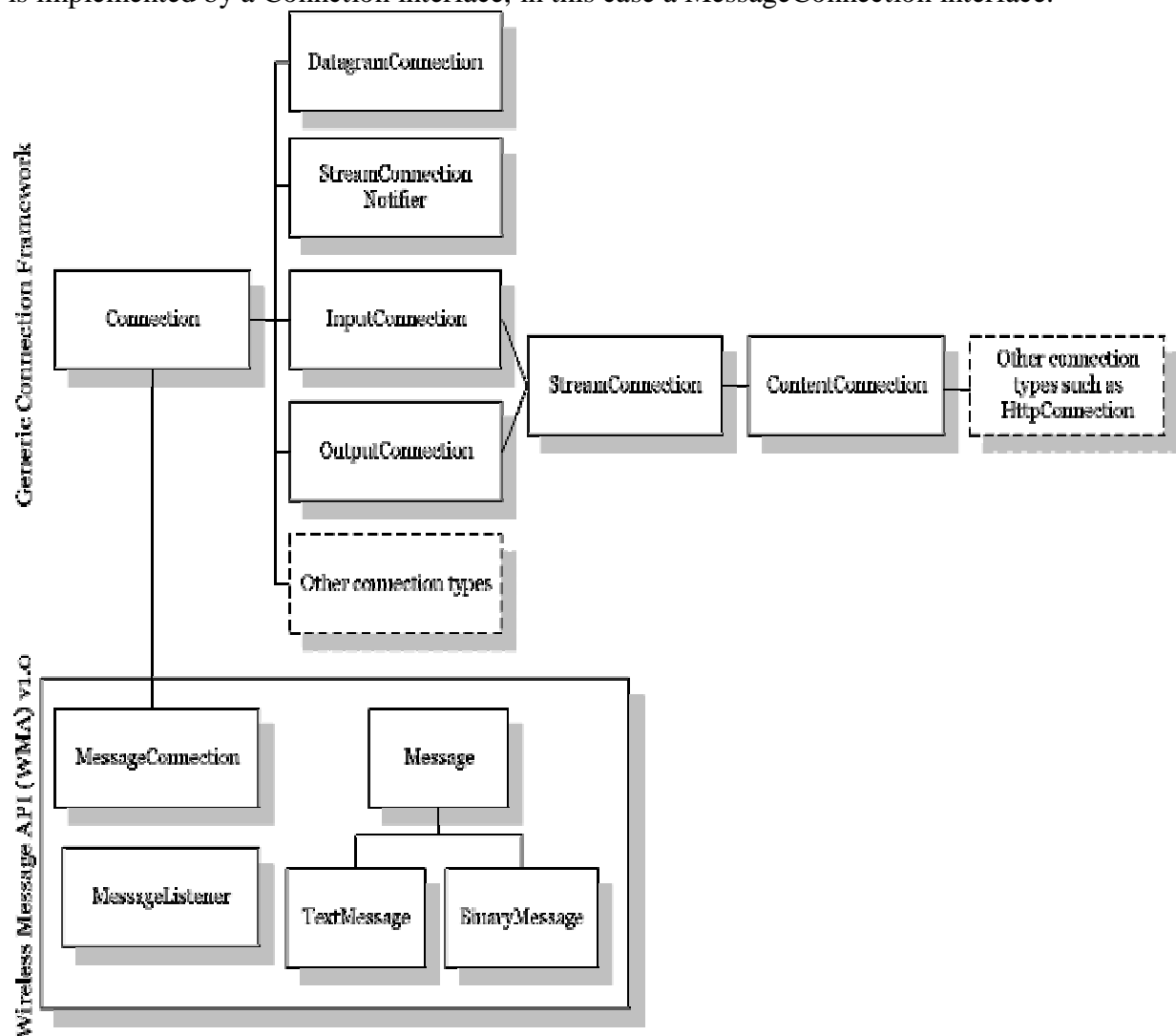
[SMS - Short and](http://developers.sun.com/techtoc/mobility/midp/articles/sms/)

[Sweet](http://developers.sun.com/techtoc/mobility/midp/articles/sms/) <http://developers.sun.com/techtoc/mobility/midp/articles/sms/>

11.1 - Describe the WMA's basic support for sending and receiving messages, and the Generic Connection Framework.

Generic Connection Framework

As defined by Generic Connection Framework the sending and receiving functionality is implemented by a Connection interface, in this case a MessageConnection interface.



Representation of a message

- A message can be thought as having an address part and a data part.
- The base interface is Message , that provides methods for addresses and timestamps.
- To handle both text and bynary messages there are two subinterfaces of message: TextMessage and BynaryMessage.

Sending and Receiving

- The application must obtain a MessageConnection from the Connector factory.
- If the string specifies a full destination address the MessageConnection acts as a client.
- If the string specifies only the protocol and port the MessageConnection acts as client/server connection. The Message object received on the other endpoint can be used for send a message for the sender since the address in the Message object will be the sender's address.
- Ever create a Message object using the new operator. For sending a Message use the factory method of the MessageConnection, MessageConnection.newMessage(String type). For receiving MessageConnection supports a event listener-based receive mechanis,. in addition to a receive() blocking method.
- For sending/receive the application MUST be granted a permission to perform the requested operation.

Codes examples (Exceptions ommited):

Sending

```
MessageConnection conn = Connector.open("sms://+358401234567:5432");
TextMessage msg = (TextMessage)
conn.newMessage(MessageConnection.TEXT_MESSAGE);
msg.setPayloadText("Hello");
conn.send(msg);
```

Receiving and Replying

```
MessageConnection conn = Connector.open("sms://:5432"); //Client/server connection
Message msg = conn.receive(); //Remember, msg contains the sender's address, this
methods blocks
if(msg instaceof MessageText){
    MessageText mReply = (MessageText) msg;
    if(mReply.getPayloadText().equals("Hello")){
        mReply.setPayloadText("Hello too!");
        conn.send(mReply);
    }
}
```

11.2 - Explain the WMA's support for SMS and Cell Broadcast capabilities.

SMS

- The GSM SMS messages consists of a fixed header and a field called TP-User-Data. The TP-User-Data field carries the payload of the short message and optional header information that is not part of the fixed header. The TP-User-Data can use different encodings on the type of the payload content. Possible encondings are a 7-bit alphabet, 8-bit binary data or 16-bit UCS-2 alphabet.

Message Payload Length

- The maximum length of the SMS protocol message payload depends on the encoding and if there are optional headers in TP-User-Data. If the optional header payload defines a port then the payload length will be smaller.
- If the Java app sends a message that is too long to fit in one SMS message, the implementation MUST use concatenation of multiple SMS messages.
- When receiving the implementation MUST automatically concatenate the received messages and pass the fully reassembled payload to the application.

Message Payload Concatenation

- Implementations MUST support at least 3 SMS protocol messages to be received and concatenated together. Similarly, for sending, messages that can be sent with up to 3 SMS must be supported.
- The MessageConnection.numberOfSegments method allows the application to check how many messages a given message will use when sent.

Message Addressing

Valid syntax are:

sms://3456789

sms://3456789:5678

sms://:5678

Specifying the recipient address

- When the port number is present in the address, the TP-User-Data MUST contain a User-Data-Header with the app port addressing. If there isn't a port in the address the TP-User-Data MUST NOT contain the Application port addressing header.

Client Mode and Server Mode Connections

- Client connections do not contain a port to reply. Only the server mode MessageConnection can be used for receiving messages.

Handling Received Messages

- When SMS messages are received by an application, they are removed from the SIM/ME memory where they have been stored.
- The GSM SMS protocol does not guarantee to preserve the ordering when multiple messages are sent.

Short Message Service Center Address

- The SMSC address used for sending messages MUST be made available using System.getProperty with the key wireless.messaging.sms.smsc.

Using Port Numbers

- The first application to allocate a given port number will get it. If another application tries to allocate the same port an IOException will be thrown.

Message Type

TextMessage: TP-Data-Coding-Scheme MUST indicate the GSM default 7-bit alphabet or USC-2. USC-2 is used if the application contains at least one character that is not present in 7-bit.

BinaryMessage: TP-Data-Coding-Scheme MUST indicate 8-bit data.

If a application try to send a message with the payload too long the MessageConnection.send() will throw an IllegalArgumentException and the message will not be sent.

Restrictions on Port Numbers for SMS Messages

- Java applications will throw a SecurityException is any of the following ports are used to send a message:

2805, 2923, 2948, 2949, 5502, 5503, 5508, 5511, 9200, 9201, 9202, 9203, 9207, 49996, 49999

Cell Broadcast

- Unidirectional data service where messages are broadcast by a base station and received by every mobile station listening to that base station.

- ONLY RECEIVE.

- Use the same encoding of SMS.

- The method Message.getTimeStamp returns always null.

Addressing

cbs://:5678

Attempts to call the send method in Cell Broadcast will throw an IOException.

11.3 - Identify correct and incorrect statements or examples about WMA including the WMA addressing scheme, client vs. server connections, WMA-related exceptions, WMA-related security issues, message size limitation, message creation, sending, synchronous vs. asynchronous message receipt, and the relationship between WMA and Push Registry.

WMA addressing scheme, client vs. server connection

Message Type	Connection String
SMS client	sms://phone[:port]
SMS server	sms://:port
CBS	cbs://port

- Remember: sending a message using a address without the port will send the message for the default mechanism handler. If a port is specified a application will handle the message.

- Client connection only SEND messages. Call to receive() in client connections will throw a IOException.

- Server connection can be used for receiving / replying messages.

WMA permissions

For opening connections: javax.microedition.io.Connector.sms and javax.microedition.io.Connector.cbs

For send and receive messages: javax.wireless.messaging.sms.send, javax.wireless.messaging.sms.receive, javax.wireless.messaging.cbs.receive

The application MUST NOT assume that successfully sending one message implies that they have the permission to send all kinds of messages to all addresses.

WMA and Push Registry

- MUST request permission to use PushRegistry and the permission to open the connection.
- If read or write data, must also request permissions.
- Filter: for sms is matched against the MSISDN (phone number) part, ignore port. for cbs filtering is not performed, is ingored.
- The SMS protocol includes an explicit buffering mechanism where messages are held until processed by some application that reads and deletes messages when they are done with data.
- When the application is started in response to a Push message, the application SHOULD read and process all messages thar are buffered for it.
- WMA includes a MessageListener which provides asynchronous callbacks when messages become available while the application is running.