

Sun Certified Mobile Application Developer (311-110)
Sun Microsystems, Inc.

Exam Notes
Sathya Srinivasan

02/09/2004

Table of Contents

Generic Connection Framework.....	3
Sending a Message.....	3
Receiving a Message.....	4
WMA Support for SMS (Short Message Service)	5
WMA Support for CBS (Cell Broadcast Service).....	5
WMA Addressing Scheme	6
Client vs. Server connections.....	6
WMA-related exceptions	6
WMA-related security issues	7
Message size limitation	7
Creating messages.....	7
Sending messages.....	8
Synchronous vs. Asynchronous message receipt	8
Relationship between WMA and Push Registry	8

Chapter 11 Wireless Messaging API (WMA) 1.1

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

Generic Connection Framework

- The `MessageConnection` interface which can be used for sending and receiving Messages derives from the `java.io.Connection` interface.
- An implementation of the `MessageConnection` interface can be obtained from the `Connector` factory, which returns a `Connection` interface and then typecast to `MessageConnection` interface.
- The messaging functionality is similar to the datagram functionality of the GCF in that
 - It allows opening a connection using a string address
 - The connection can be opened in *client* or *server* mode.
- The `MessageConnection` SHOULD be closed using the `close()` method.
- If a `MessageConnection.newMessage(...)` method is called after closing the connection, the return value will be an implementation of the interface `Message` and NOT a sub-interface, even if asked for.
- The `MessageConnection` DOES NOT support the *stream* methods. If invoked, it will throw an `IllegalArgumentException`.

Sending a Message

- A `MessageConnection` implementation is obtained by calling `Connector.open(String name)` method and typecasting the return value to a `MessageConnection`.
- For sending a message, the name should be of the form "`scheme://target`". Typically this would be something like "`sms://+1234567890`".
 - The schema name (here, "sms") can be anything, as defined by the mobile device implementation.
 - In this case, the Connection is said to be in the client mode.
 - The number represents the phone number where the message is to be sent.
 - The `Connector.open(String name, int mode)` can be used to represent the mode of connection. The mode can be `READ`, `WRITE`, or `READ_WRITE`.
 - If an attempt is made to open a connection in `READ` mode where it is not possible to read anything (say, a printer), then it will throw an `IllegalArgumentException`.
 - The `Connector.open(String name, int mode, int timeouts)` can be used if the underlying implementation supports timeouts. If a timeout occurs, an `InterruptedIOException` will be thrown back.
- The `Message` to be sent SHOULD be created with the same instance of the `MessageConnection` interface using the `MessageConnection.newMessage(String type, String address)` method.
 - The type can be `MessageConnection.TEXT` (or the String "text") or `MessageConnection.BINARY` (or the String "binary").

- The type is case-sensitive.
- There can be other types supported by the mobile device. But they should have the format similar to package name convention and should contain at least one '.'
- The `address` parameter is optional. If not mentioned, the address used to create the `MessageConnection` is used.
- An `IllegalArgumentException` can be thrown if
 - The `Message` to be sent is not in the right format
 - Does not contain valid information
 - The type is not correct
 - The payload exceeds the maximum allowed length.
 - The `Message` is not of the right type (this can happen if a different `MessageConnection` is used than the one used to create the `Message`).
- An `InterruptedIOException` is thrown if the connection is closed before the `Message` is successfully sent.
- A `NullPointerException` is thrown if the `Message` is null.
- `Messages` can be sent in *client* mode as well as *server* mode. In the *server* mode, the destination address will be the same as the origin address.

Receiving a Message

- An application can listen for messages in the *server* mode.
- For receiving a message, the name should be of the form "`scheme://:port`". Typically this would be something like "`sms://:5432`".
- If an attempt is made to start listening on a port already reserved or in use, the `open()` method will throw an `IOException`.
- A `Message` can be received by invoking the `MessageConnection.receive()` method. This is a blocking call and will wait till a `Message` comes to this application.
- An instance of `MessageListener` can be used to register to listen for incoming `Messages`. When a `Message` comes to the port where the application is listening, the `notifyIncomingMessage()` method will be invoked.
- A `MessageListener` instance can be registered to the `MessageConnection` using the `setMessageListener(MessageListener listener)` method.
- The application has the responsibility to receive the message using the `MessageConnection.receive()` method once notified.
- The `notifyIncomingMessage()` should be a quickly returning method and the `Message` SHOULD NOT be received in the same thread. A separate thread should be used to invoke the `receive()` method.
- The implementation MAY call the listener from multiple threads for multiple messages that arrive closely. The application should *synchronize* to handle this.
- An `IOException` can occur if the `receive()` method is called when the connection was opened in the *client* mode.
- The application is responsible for persisting a received message (say, in the RMS) if needed.

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

WMA Support for SMS (Short Message Service)

- SMS is supported both in GSM and CDMA network.
- `TextMessages` should be encoded in GSM-7 bit or UCS-2 16 bit encodings. If a character is present in the message that is not in GSM-7 bit encoding, the message should be automatically treated as UCS-2 16 bit encoding.
- If a port number is not mentioned in the address of a message being sent, it will be sent to the device. If the port number is present, it will be sent to the application in the device.
- Long Messages can be split and concatenated. A minimum of 3 splits should be supported by the device.
- It is advised to limit the splits to 3. The number of splits of a given Message can be found out by calling the `numberOfSegments()` method.
- SMS messages can be sent or received. The standard WMA rules apply with regard to *client* and *server* modes.
- Some ports are restricted and if an attempt is made to use them, a `SecurityException` will be thrown.
- The Short Message Service Center (SMSC) address should be available in the System property `wireless.messaging.sms.smsc`. This is a phone number in the `msisdn` format.

WMA Support for CBS (Cell Broadcast Service)

- In a CBS, the base station of the service provider can broadcast a message, which can be viewed by all the subscribers to that base station.
- The address of a CBS message takes the form "`cbs://port`", where `port` is the port number where the message will be received in the application.
- The Message can be `BinaryMessage` or `TextMessage`, but can only be received by an application and cannot be sent. If the `send()` method is called with a CBS address, `IOException` will be thrown.

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

Message Type	Address Format
GSM/CDMA SMS in client mode	<code>sms://msisdn[:port]</code>
GSM/CDMA SMS in server mode	<code>sms://:port</code>
GSM CBS (only server mode)	<code>cbs://:port</code>

- `msisdn` is phone number of the format +1234567890
- `port` is a simple port number
- If the port is not given in the SMS client mode, the message will be sent to the device's default message handling mechanism.
- If the port is given, then the message will be sent to the Java application meant to receive it.

Client vs. Server connections

- Client connections can be used to send messages only. If the `receive()` method is called, it will throw an `IOException`.
- Server connections can be used to send and receive messages. If a server connection is used to send a message, it can reuse the address present in the message received from that connection.

WMA-related exceptions

Exception	Reason for throwing the exception
<code>IllegalArgumentException</code>	Message to be sent is too long. Connection type is not supported for the operation. Message to be sent is invalid or incorrect.
<code>InterruptedException</code>	Connection was closed while sending or receiving data. Timeout is caused if timeout is specified during <code>open()</code> call.
<code>IOException</code>	<code>receive()</code> method is called in client mode.
<code>NullPointerException</code>	Message to be sent is null.
<code>SecurityException</code>	<code>send()</code> or <code>receive()</code> message is not permitted. A restricted or reserved port was attempted to be used as receiving port.

WMA-related security issues

- As sending and receiving SMS messages may incur a charge, user should be asked for permission before attempting to send or receive a message.
- Permission MAY depend on the type of message and the address used.
- The application MAY be restricted from sending/receiving certain types of messages or to certain addresses.
- Permission to send ONE message DOES NOT mean that successive messages will be granted permission.
- `SecurityException` may be thrown if an operation is attempted without proper permission.
- Granting permissions on devices with MIDP 1.0 is left to the user.
- Permissions on granted on devices implementing MIDP 2.0 based on protocols.

Message size limitation

Header encoding	Segments	Maximum Message Length (bytes)	
		With Port Number	Without Port Number
GSM 7-bit text	1	160	152
	2	304	290
	3	456	435
8-bit binary data	1	140	133
	2	266	254
	3	399	381
UCS-2 text	1	70	66
	2	132	126
	3	198	189

Creating messages

- The `Message` to be sent SHOULD be created with the same instance of the `MessageConnection` interface using the `MessageConnection.newMessage(String type, String address)` method.
 - The `type` can be `MessageConnection.TEXT` (or the String "text") or `MessageConnection.BINARY` (or the String "binary").
 - The `type` is case-sensitive.
 - There can be other types supported by the mobile device. But they should have the format similar to package name convention and should contain at least one '.'
 - The `address` parameter is optional. If not mentioned, the address used to create the `MessageConnection` is used.
- An `IllegalArgumentException` can be thrown if
 - The `Message` to be sent is not in the right format
 - Does not contain valid information
 - The `type` is not correct
 - The payload exceeds the maximum allowed length.
 - The `Message` is not of the right type (this can happen if a different `MessageConnection` is used than the one used to create the `Message`).

Sending messages

See [Sending a message](#).

Synchronous vs. Asynchronous message receipt

- Messages can be received synchronously using the `MessageConnection.receive()` method. This is a blocking call and will not return till a message is received, even though it may take years!
- Messages can be received asynchronously by creating a class implementing the `MessageListener` interface and registering an instance of the class to the connection using the `MessageConnection.setMessageListener()` method.
 - The `notifyIncomingMessage()` method of the `MessageListener` instance is invoked when a message is received.
 - The `notifyIncomingMessage()` implementation should return very quickly and SHOULD NOT get the message by calling `MessageConnection.receive()`. This should be done in a separate thread.
 - If many messages come in a short period of time, the VM may notify the application of the messages in multiple simultaneous threads. The application should handle the invocations in a synchronous manner.

Relationship between WMA and Push Registry

- WMA applications make use of Push Registry to launch the application if a message is received when the application is not running or when another MIDlet is running.
- Permission should be obtained by the application to use the Push Registry apart from the standard permissions to open connections, and for read/write.
- Permission entry for push registrations related to WMA is for the form
 - `MIDlet-Push-n: protocol (sms | cbs)://:port, classname, address`
 - The last entry indicates the source address of the message for which permission is granted. For all addresses, the wildcard `*` maybe used.
 - For `cbs` connections, the `address` is always `*`.
- All successfully received messages will be buffered till they are read and deleted by the application.
- If the application DOES NOT read and delete messages upon startup, the device MAY delete the messages if it becomes necessary (say, buffer is full).