

Sun Certified Mobile Application Developer (311-110)

Sun Microsystems, Inc.

Exam Notes

Sathya Srinivasan

02/18/2004

Table of Contents

Generic Connetion Framework.....	3
GCF Interfaces.....	4
MIDP Characteristics.....	4
Class Hierarchy Relationships.....	6
Protocols supported by MIDP 2.0.....	7
Mandatory Connections.....	7
Optional Connections.....	7
Issues with TCP/IP and Datagram Connections.....	7
Opening a networking connection.....	8
Closing a connection.....	8

Chapter 4 Networking

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.

Generic Connection Framework

The GCF has been designed in such a way to provide maximum flexibility to its implementation while preserving uniformity. The following steps are typically followed to setup a connection using GCF classes.

- Create a URL to connect to. This a `String` of the form `[schema]:[host][:port]`
- Use the factory class `Connector` to create a connection
 - `Connector.open(String url, {int mode}, {int timeout}):Connection`
 - The optimal mode parameter can be `READ`, `WRITE`, or `READ_WRITE`
 - The default mode is `READ_WRITE`
 - If an unsupported mode is used, an `IllegalArgumentException` will be thrown.
 - Timeout can be used to specify that the caller wants to know about timeouts. If `true`, the protocol may throw an `InterruptedException` when timeout occurs. This is only a suggestion to the implementation and is not guaranteed.
- Cast the return value (a `Connection`) to the appropriate sub-interface of the `Connection`. The `Connector` framework will create an appropriate sub-interface based on the protocol given in the passed URL.
- Use the following methods to get the streams for the `Connection`.
 - `Connection.getInputStream()`
 - `Connection.getOutputStream()`
 - `Connection.getDataInputStream()`
 - `Connection.getDataOutputStream()`
- Sub-interface specific methods can be used to get further information from the stream.
- Use the `Connection.close()` method to close the stream.
- Once an `InputStream` or `OutputStream` (or its `Data` counterparts) is obtained, attempts to get another one of the same type will throw an `IOException`. For example, you cannot call `Connector.openInputStream()` twice.

GCF Interfaces

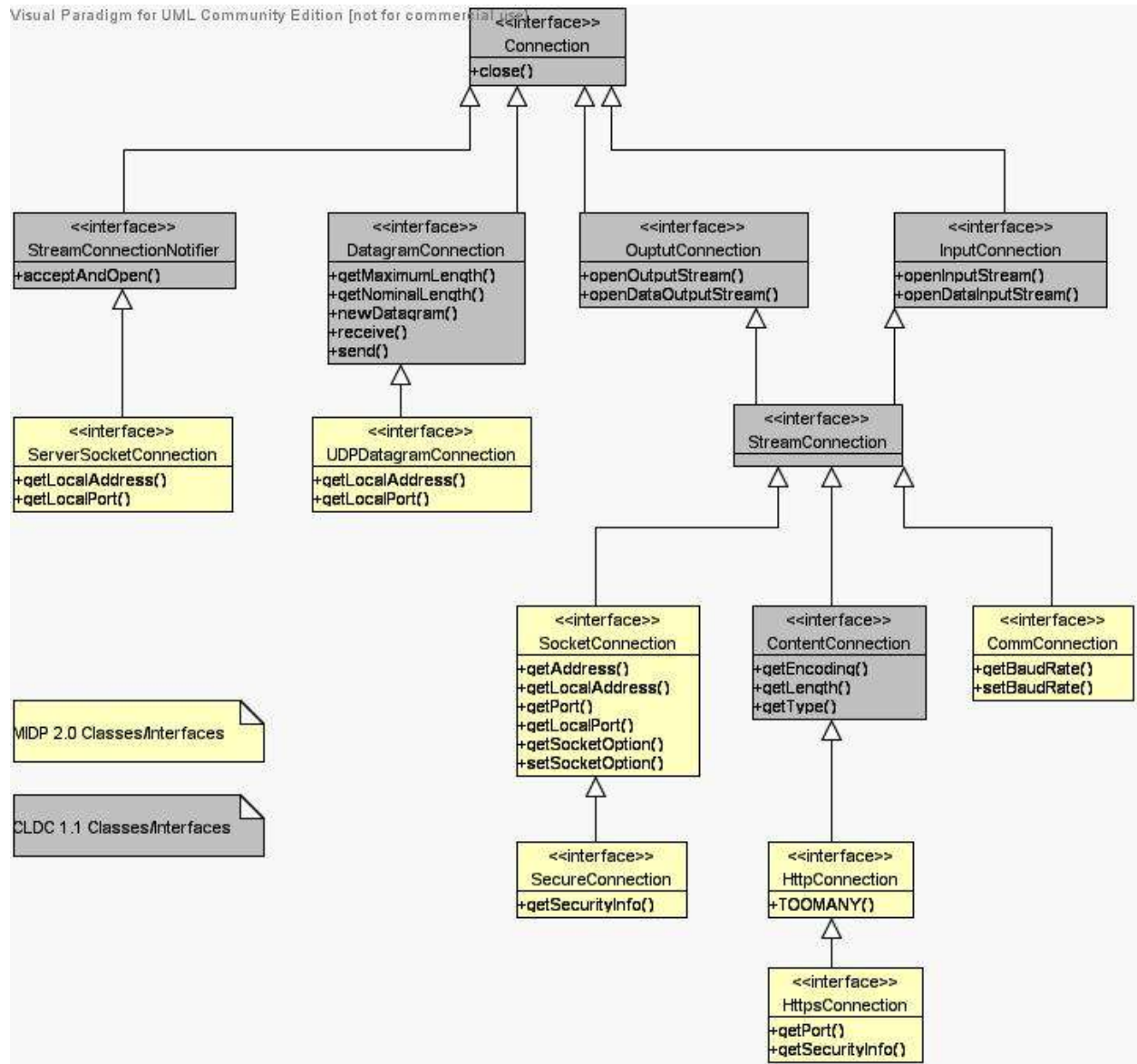
- The CLDC **DOES NOT** support any network-specific protocol. Only generic interfaces are provided by the CLDC which are expected to be sub-classed (or sub-interfaced) by the profiles.
- The URL for a connection should be of the format `[protocol]:[host][:port][;parameters]`
- `Connector`
 - A factory to provide `Connection` interface implementations.
 - The `Connector` creates the suitable implementation based on protocol portion of the URL passed in its `open()` method. The protocol portion is the substring of the URL from the beginning to the first occurring colon (:).
 - Convenience methods for opening connection streams are also present. These have the same behavior as their counterparts in `Connection` sub-interfaces.
- `Connection`
 - The super-interface that supports opening input and output streams and closing the connection.
 - The `open()` method is not provided here as the `Connector` class should be used to create and open a connection.
- `InputConnection` (Serial input connection)
 - Implemented by devices from where data can be read. Provides the `openInputStream()` and `openDataInputStream()` methods.
- `OutputConnection` (Serial output connection)
 - Implemented by devices to which data can be written. Provides the `openOutputStream()` and `openDataOutputStream()` methods.
- `StreamConnection` (Circuit connection)
 - Implemented by devices that provide two-way communication. This interface combines the `InputConnection` and `OutputConnection` interfaces.
- `ContentConnection` (Content-based connection like a web server connection)
 - This is a sub-interface of the `StreamConnection` that provides extra methods for getting content information, like `type`, `length`, and `encoding`.
- `StreamConnectionNotifier` (Server portion of a client-server connection)
 - Implemented by server connections. This can be used to accept connections and notify a class of the incoming connection. The `acceptAndOpen()` method is defined here.

MIDP Characteristics

- The baud rate specified in the `CommConnection` is only a suggestion. The device may set whatever is possible. The actual baud rate can be obtained using the `CommConnection.getBaudRate()` method.
- If no host or port is given in the URL of a `SocketConnection` or a `ServerSocketConnection` (`socket://`), then the device will assign the next available free port. This can be obtained using the `getLocalPort()` method. The host name will also be assigned automatically. This will typically be the system property `"microedition.hostname"` if available.
- If the host portion is not specified in a datagram URL, then the connection is treated as a server connection (that is, it can receive Datagrams). If the host portion is specified, the connection is treated as a client connection (that is, it can send Datagrams).
 - In the server mode, the port is the port where data is received. This is the port for both sending and receiving data.

- In the client mode, the port is the port in the target system where the data is to be sent. The reply-to port is dynamically allocated.
- The `Datagram.receive()` is a blocking call.
- Length of a Datagram
 - When the Datagram is being sent, this represents the length of data to be sent.
 - Before receiving the Datagram, this represents the length of bytes to receive.
 - After receiving the Datagram, this represents the length of bytes received.
- The `Datagram.reset()` method resets the data of the Datagram (length, offset, and buffer).

Class Hierarchy Relationships



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.

Protocols supported by MIDP 2.0

URL Format	Protocol	Connection Subclass
<code>http://host[:port][params]</code>	HTTP 1.1	HttpConnection
<code>https://host[:port][params]</code>	HTTPS	HttpsConnection
<code>ssl://host[:port][params]</code>	Secure Socket Layer (SSL)	SslConnection
<code>socket://[host][:port]</code>	Socket	If host is given, this returns a SocketConnection. If only port is given, this returns a ServerSocketConnection. If both are not given, this returns a SocketConnection with a default value for host and port.
<code>datagram://[host][:port]</code>	UDP	UdpDatagramConnection
<code>comm:[device][params]</code>	Communication Devices (Ex. RS-232 Serial, IrDA)	CommConnection

Mandatory Connections

- HttpConnection
- HttpsConnection (not sure)

Optional Connections

- UdpDatagramConnection
- SocketConnection
- ServerSocketConnection
- CommConnection (not sure)
- SecureConnection (not sure)

Issues with TCP/IP and Datagram Connections

- Delivery of data and duplicate protection is not guaranteed in UDP. Delivery is guaranteed in TCP/IP.
- The destination address is specified inside the Datagram for a UDP connection. It is specified in the connection for a TCP/IP connection.

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.

Opening a networking connection

- A connection can be created in one of the following ways.
 - `Connection connection = Connector.open("protocol://host:port;paramList");`
 - `Connection connection = Connector.open("protocol://host:port;paramList", Connector.READ);`
 - `Connection connection = Connector.open("protocol://host:port;paramList", Connector.WRITE, true);`
 - `InputStream in = Connector.openInputStream("protocol://host:port;paramList");`
 - `DataInputStream dataIn = Connector.openDataInputStream("protocol://host:port;paramList");`
 - `OutputStream out = Connector.openOutputStream("protocol://host:port;paramList");`
 - `DataOutputStream dataOut = Connector.openDataOutputStream("protocol://host:port;paramList");`
- Typically, the connection returned (in the first 3 options) is typecast into the appropriate sub-interface to get more information from the connection.
- Examples of valid URLs

URL	Protocol Type	Connection Interface
<code>comm:com0;baudrate=19200</code>	Serial connection protocol, typically to an RS-232 or IrDA port.	<code>CommConnection</code>
http://www.cssathya.com:80	HTTP protocol	<code>HttpConnection</code>
<code>https://www.verisign.com:8080</code>	Secure HTTP protocol (HTTPS)	<code>HttpsConnection</code>
<code>socket://</code> or <code>socket://:port</code>	Custom server socket	<code>ServerSocketConnection</code>
<code>socket://host:port</code>	Either a socket or a server socket	<code>SocketConnection</code> or <code>ServerSocketConnection</code>
<code>ssl://host:port</code>		<code>SecureConnection</code>

Closing a connection

- Connections can be closed using the `Connection.close()` method.
 - If a `Connection` is already closed, then calling this method has no effect.
 - If the associated streams are open, then this method will wait till they are closed.
 - In this case, the `Stream` objects can be accessed, but accessing a method of the `Connection` interface (or its sub-interface) will throw `IOException`.
 - This method is thread-safe.

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

Exception	Method	Reason
IOException	All open() and close() methods and their variations.	I/O error while attempting to open or close a connection or while attempting to obtain a connection's stream.
SecurityException	All Connector.open() methods and their variations.	The application is not allowed to open a connection.
ConnectionNotFoundException	All Connector.open() methods and their variations.	The target could not be found or if the requested protocol is not supported.
InterruptedIOException	Connector.open() Datagram.receive()	If timeout parameter is set to true in a Connector.open() method and connection times out or if the connection is lost during a DatagramConnection.receive() call.
EOFException	XXXStream.read()	End-of-File is reached.