Bab 6
Jaringan

Pada bagian ini, kita akan belajar bagaimana menerapkan sebuah MIDlet yang mempunyai kemampuan untuk koneksi kedalam jaringan.

Pada bagian akhir dari sesi ini, siswa diharapkan dapat:

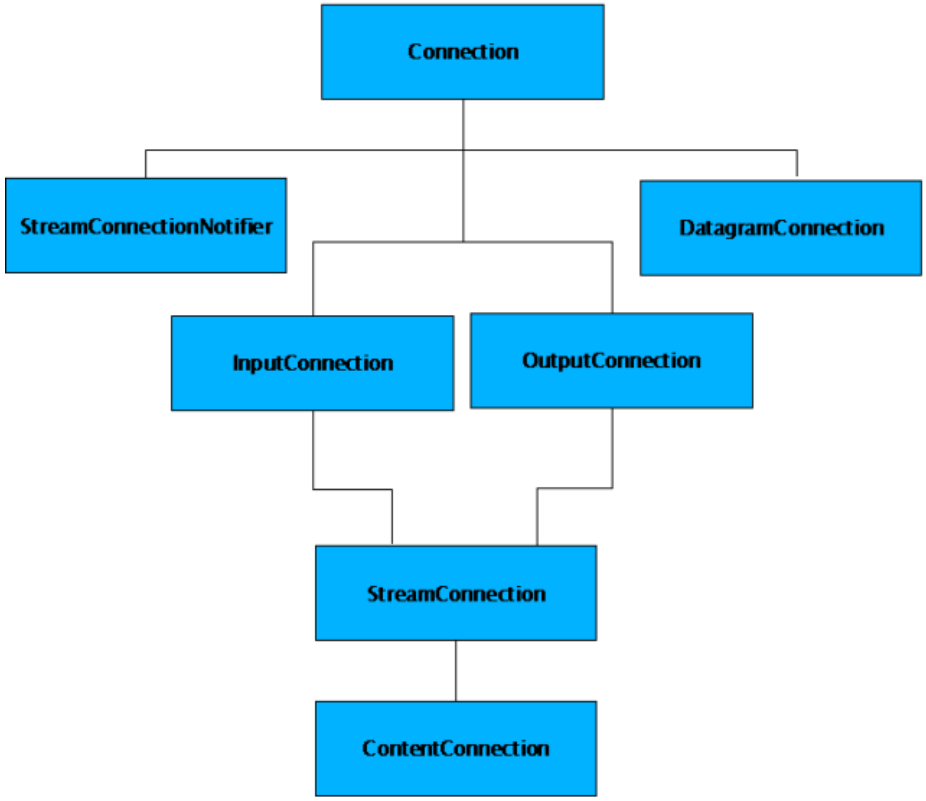
- Mendeskripsikan Generic Connection Framework, dan bagaimana ia dapat digunakan untuk mendukung method koneksi yang berbeda-beda.
- Menspesifikasikan parameter-parameter koneksi dengan menggunakan format pengalamatan GCF URL
- Membuat koneksi HTTP/HTTPS
- Menciptakan MIDlet dengan menggunakan TCP sockets dan UDP datagram

6.1 Generic Connection Framework

Generic Connection Framework mendukung koneksi packet (socket) dan stream (datagram). Sesuai dengan namanya, framework ini menyediakan API dasar bagi koneksi di CLDC. Framework ini menyediakan pondasi umum dari berbagai koneksi seperti HTTP, socket, dan datagram. Walaupun Bluetooth dan serial I/O termasuk kedalam API ini, GCF menyediakan satu set API yang lebih generic dan mendasar yang menjadi abstraksi dari berbagai tipe koneksi. Harus dicatat, bahwa tidak semua tipe koneksi dibutuhkan bagi implementasi sebuah MIDP device.

6.1.1 Hirarki dari GCF Interface

Perluasan dari hirarki GCF interface memungkinkan terjadinya generalization. Sebuah tipe koneksi yang baru mungkin dapat ditambahkan kepada framework ini dengan cara memperluas hirarki.



Gambar 8.1: Hirarki dari GCF Interface

6.1.2 GCF Connection URL

Parameter-parameter koneksi telah dispesifikasikan dengan menggunakan sebuah format pengalamatan:

scheme://username:password@host:port/path;parameters

1. Scheme adalah sebuah protokol atau method koneksi. Misalnya: http,ftp, https.
2. Username bersifat optional, akan tetapi bila kita ingin mendefinisikannya, harus didahului dengan tanda @
3. Password juga bersifat optional dan hanya dapat dispesifikasikan jika username telah didefinisikan sebelumnya. Jika password didefinisikan, maka ia harus dipisahkan dari username dengan menggunakan tanda titik dua (:)
4. Host – parameter ini wajib dicantumkan. Bisa berupa nama host atau fully qualified domain name (FQDN) atau alamat IP dari host yang dituju.
5. Port – parameter ini juga bersifat optional. Jika tidak dispesifikasikan, maka default port akan digunakan
6. Path
7. parameters – bersifat optional, tetapi harus didahuli dengan titik koma (;) apabila ia dicantumkan

Jika kita menggunakan kurung siku untuk memberi tanda pada parameter-parameter yang bersifat optional pada format pengalamatan diatas, kita dapat mengubah format diatas menjadi seperti berikut:

scheme://[username[:password]@]host[:port]/path[;parameters]

Format pengalamatan tersebut haruslah sesuai dengan Uniform Resource Indicator (URI) seperti yang didefinisikan pada RFC 2396.

Pada MIDP 2.0, hanya skema “http” dan “https” dibutuhkan untuk diimplementasikan pada device.

6.2 Koneksi HTTP

6.2.1 Protokol HTTP

HTTP merupakan kepanjangan dari HyperText Transfer Protocol. Ia merupakan protocol yang digunakan untuk memindahkan web pages dari web server (misal: www.sun.com) kepada web browser. Client(web browser) akan me-request sebuah web page dengan cara mespesifikasikan path dengan command Get atau POST.

Pada method GET, parameter telah dispesifikasikan dan dilekatkan pada URL. Sebagai contoh, untuk memberikan sebuah variable dengan nama "id" dan memiliki nilai 100 kepada index.jsp, url tersebut akan dispesifikasikan sebagai : "http://hostname/index.jsp?id=100". Parameter tambahan dipisahkan dengan dengan tanda &, "http://hostname/index.jsp?id=100&page=2.

Jika method "POST" digunakan, parameter bukanlah menjadi bagian dari URL tetapi dikirim dengan pada baris terpisah pada command POST.

Client / Web Browser	HTTP Server
GET /index.jsp?id=100 HTTP/1.1	
	HTTP/1.1 200 OK Server: Apache-Coyote/1.1 Content-Type: text/html;charset=ISO-8859-1 Date: Wed, 18 Jun 2005 14:09:31 GMT Connection: close <html> <head> <title>Test Page</title> </head> <body> Test Page </body> </html>

Gambar 8.2: Contoh dari transaksi HTTP GET

Client / Web Browser	HTTP Server
GET /non-existent.html HTTP/1.0	
	HTTP/1.1 404 /non-existent.html Server: Apache-Coyote/1.1 Content-Type: text/html;charset=utf-8 Content-Length: 983 Date: Mon, 11 Jul 2005 13:21:01 GMT Connection: close <html><head><title>Apache Tomcat/5.5.7 - Error report</title><style>... <body> HTTP Status 404 ... The requested resource (non-existent.html) is not available. ... </body></html>

Gambar 8.3: Contoh dari transaksi HTTP GET dengan response error

6.2.2 Menciptakan sebuah koneksi HTTP

Anda dapat membuka sebuah koneksi HTTP dengan menggunakan Connector.open() dan meng-casting nya dengan salah satu dari ketiga interface berikut ini: StreamConnection, ContentConnection, dah HTTPConnection. Bagaimanapun, dengan StreamConnection dan ContentConnection, Anda tidak dapat menspesifikasikan dan menurunkan parameter-parameter spesifik dari HTTP dan juga result-nya.

Bila Anda menggunakan StreamConnection, panjang dari sebuah reply, tidak dapat ditentukan sebelumnya. Sedangkan pada ContentConnection atau HTTPConnection, selalu ada cara untuk menentukan panjang dari sebuah reply. Akan tetapi penentuan panjang ini, tidak selalu tersedia. Oleh karena itu, program Anda harus bisa mendapatkan reply tersebut tanpa harus mengetahui panjang content terlebih dahulu.

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

HttpConnection connection = null;
InputStream iStream = null;
byte[] data = null;

try {
    connection = (HttpConnection) Connector.open("http://www.sun.com/");
    int code = connection.getResponseCode();

    switch (code){
        case HttpConnection.HTTP_OK:
            iStream = connection.openInputStream();
            int length = (int) connection.getLength();
            if (length > 0){
                data = new byte[length];
                int totalBytes = 0;
                int bytesRead = 0;
                while ((totalBytes < length) && (bytesRead > 0)) {
                    bytesRead = iStream.read(
                        data, totalBytes, length - totalBytes);
                    if (bytesRead > 0){
                        totalBytes += bytesRead;
                    }
                }
            } else {
                   //panjang tidak diketahui, baca tiap karakter
                ...
            }
            break;
        default:
            break;
    }

...
```

6.2.3 Handling HTTP Redirects

Terkadang server akan melakukan redirect dari sebuah browser/client ke web page yang lain dengan cara me-reply HTTP_MOVED_PERM (301), HTTP_MOVED_TEMP (302), HTTP_SEE_OTHER (303) atau HTTP_TEMP_REDIRECT (307) daripada menggunakan reply HTTP_OK yang biasa dilakukan. Program Anda harus dapat mengidentifikasi-nya dengan menggunakan getResponseCose(), mendapatkan URI yang baru dari header dengan menggunakan getHeaderField("Location"), dan mendapatkan kembali dokumen dari lokasi yang baru.

```
int code = connection.getResponseCode();

switch(code){
    case HttpConnection.HTTP_MOVED_PERM:
    case HttpConnection.HTTP_MOVED_TEMP:
    case HttpConnection.HTTP_SEE_OTHER:
    case HttpConnection.HTTP_TEMP_REDIRECT:
        String newUrl = conn.getHeaderField("Location");

...
```

6.3 Koneksi HTTPS

HTTPS adalah sebuah HTTP diatas sebuah koneksi secure transport. Membuka sebuah koneksi HTTPS, hampir sama untuk membuka koneksi HTTP. Perbedaan utamanya adalah URL akan memberikan kepada Connector.open() dan meng-casting hasilnya kepada HttpsConnection class variable.

Sebuah tipe exception tambahan juga harus dijalankan melalui Connector.open() misalnya **IllegalArgumentException**, **ConnectionNotFoundException**, **java.io.IOException** dan **SecurityException**. Sebuah **CertificateException** juga dapat dijalankan untuk melaporkan kesalahan pada certificate.

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

HttpsConnection connection = null;
InputStream iStream = null;
byte[] data = null;


try {
    connection = (HttpsConnection) Connector.open("https://www.sun.com/");
    int code = connection.getResponseCode();
    ...
} catch (CertificateException ce){
    switch (ce.getReason()){
        case CertificateException.EXPIRED:
        ...
    }
}
```

static byte	BAD_EXTENSIONS Mengindikasikan bahwa sertifikat memiliki extenstion yang tidak teridentifikasi.
static byte	BROKEN_CHAIN Mengindikasikan bahwa sertifikat terletak didalam sebuah rantai yang tidak terautentikasi pada mata rantai berikutnya.
static byte	CERTIFICATE_CHAIN_TOO_LONG Mengindikasikan bahwa sertifikat server dari rantai tersebut melebihi panjang yang disepakati pada policy dari pembuat sertifikat.
static byte	EXPIRED Mengindikasikan bahwa sertifikat tersebut telah berakhir jangka waktunya.
static byte	INAPPROPRIATE_KEY_USAGE Mengindikasikan bahwa public key dari sertifikat tersebut telah digunakan tidak sesuai dengan ketetapan yang dibuat oleh pembuat sertifikat.
static byte	MISSING_SIGNATURE Mengindikasikan bahwa object dari sertifikat tidak memiliki sebuah tanda tangan digital.
static byte	NOT_YET_VALID Mengindikasikan bahwa sertifikat tersebut tidak berlaku.
static byte	ROOT_CA_EXPIRED Mengindikasikan bahwa root dari public key CA telah habis jangka waktunya.
static byte	SITENAME_MISMATCH Indicates a certificate does not contain the correct site name.
static byte	UNAUTHORIZED_INTERMEDIATE_CA Mengindikasikan bahwa ada sebuah sertifikat intermediate certificate didalam rantai yang tidak punya otoritas sebagai intermediate CA.
static byte	UNRECOGNIZED_ISSUER Mengindikasikan bahwa sertifikat tersebut telah dikeluarkan oleh entity yang tidak teridentifikasi.
static byte	UNSUPPORTED_PUBLIC_KEY_TYPE Mengindikasikan bahwa tipe public key didalam sertifikat tidak didukung oleh device.
static byte	UNSUPPORTED_SIGALG Mengindikasikan bahwa sertifikat telah ditandatangani dengan menggunakan algorithma yang tidak disupport.
static byte	VERIFICATION_FAILED Mengindikasikan bahwa sertifikat tersebut gagal di-verifikasi.

Gambar 8.4: Berbagai alasan pada CertificateException

(kutipan dari spesifikasi MIDP 2.0 – JSR 118)

6.4 TCP Sockets

Banyak implementasi dari HTTP dijalankan diatas layer TCP. Jika Anda mengirim data menggunakan layer TCP, data tersebut akan dipotong menjadi bagian yang lebih kecil yang disebut dengan packet. Layer TCK akan memastikan bahwa semua packet akan dikirim oleh sender dan diterima oleh recipient, dengan susunan yang sama seperti pada saat ia dikirimkan. Jika sebuah packet tidak diterima oleh recipient, ia akan mengirimkannya kembali. Hal ini berarti, sekali Anda mengirim sebuah pesan, Anda dapat memastikan bahwa pesan tersebut akan berhasil dikirim kepada recipient dengan format yang sama seperti pada saat Anda mengirimkannya, tanpa ada data yang hilang atau disisipi (dihalangi oleh sebuah siklus tertentu seperti recipient *disconnect* dari jaringan) .

Layer TCP menangani reassembly dan retransmission pada packet secara transparan. Sebagai contoh, pada protokol HTTP kita tidak perlu khawatir terhadir prosess disassembly dan assembly packet karena hal ini akan dihandle pada layer TCP.

Kadang-kadang, ukuran dari pesan tersebut terkadang terlalu kecil dan sangat tidak efisien untuk dikirimkan sebagai packet tunggal (overhead dari packet sangat tinggi jika dibandingkan dengan payload). Bayangkan banyak packet dikirimkan melalui jaringan dengan satu byte payload dan multi byte overhead(misal 16 bytes). Hal ini akan menyebabkan jaringan sangat tidak effisien, banyak packets membanjiri jaringan dengan hanya satu byte payload.

Pada kasus ini, implementasi dari TCP dimungkinkan untuk menunggu sebuah pesan dikirim dengan sukses. Pesan tersebut kemudian akan dipaket sebagai banyak pesan didalam sebuah packet sebelum dikirimkan. Jika hal ini terjadi, maka akan terjadi keterlambatan pada koneksi. Jika aplikasi Anda menginginkan sesedikit mungkin terjadi keterlambatan, anda harus mengeset DELAY socket option ke nol (0). Atau jika aplikasi Anda dapat tetap berjalan dengan beberapa paket yang hilang atau tidak tersusun secara benar, Anda mungkin harus mencoba menggunakan UDP atau koneksi datagram. Koneksi UDP juga menggunakan sesedikit mungkin overhead packet.

```
SocketConnection conn =
    (SocketConnection) Connector.open("socket://www.sun.com:80");

client.setSocketOption(DELAY, 0);

InputStream iStream = conn.openInputStream();
OutputStream oStream = conn.openOutputStream();

os.write("GET / HTTP/1.0\n\n".getBytes());

int c = 0;
while((c = is.read()) != -1) {
    // memproses data yang diterima
    ...
}
```

```
iStream.close();
oStream.close();
conn.close();
```

6.5 Server Sockets

Didalam model client-server, server akan secara terus menerus menunggu sebuah koneksi dari client atau dari port tertentu yang telah disetujui. Kita juga dapat menggunakan method Connector.open untuk menciptakan sebuah server socket. Sebuah URL akan memberikan sebuah format yang sama seperti pada TCP socket kepada method open(), dengan nama hostname yang dibiarkan kosong (misal socket://:8899).

```
ServerSocketConnection conn =
    (ServerSocketConnection) Connector.open("socket://:8889");

// Dengarkan koneksi dari client
SocketConnection client = (SocketConnection) conn.acceptAndOpen();

client.setSocketOption(DELAY, 0);

InputStream iStream = client.openInputStream();
OutputStream oStream = client.openOutputStream();

// baca/tulis untuk input/output streams
...

is.close();
os.close();
client.close();
server.close();
```

6.6 Datagrams

Koneksi dari TCP socket adalah koneksi yang dapat dipercaya. Sebaliknya, tersampaikannya pesan dengan menggunakan packet UDP tidak dijamin. Tidak ada jaminan bahwa packet yang dikirimkan dengan menggunakan paket datagram akan diterima oleh pasangan. Susunan dari packet yang diterima juga tidak terpercaya. Susunan packet yang dikirimkan dimungkinkan untuk tidak sama dengan susunan packet yang diterima.

UDP datagrams atau packet digunakan apabila aplikasi dapat tetap berjalan walaupun ada packet yang hilang atau packet tersebut tidak lagi memiliki susunan yang sama seperti yang dikirimkan.

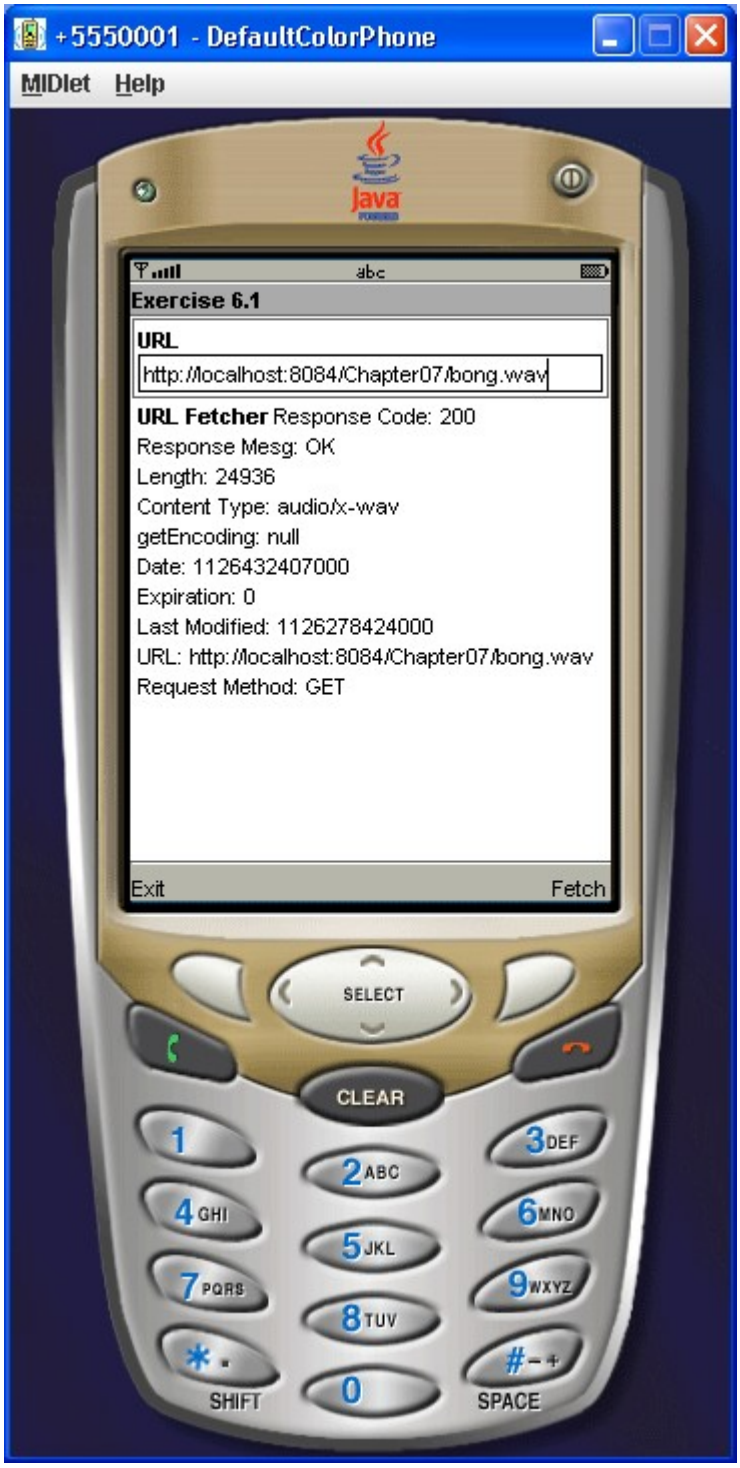
```
String url;
try {
    if (isServer){
        // memulai sebagai server, mendengarkan port 8888
        url = "datagram://:8888";
    } else {
          // memulai sebagai client, koneksi dengan port 8888 sebagai
          //localhost
        url = "datagram://localhost:8888";
    }
    dc = (DatagramConnection) Connector.open(url);

    while (true) {
        Datagram dgram = dc.newDatagram(128);
        dc.receive(dgram);
        if (isServer){
            // memulai sebagai server, mendapatkan alamat koneksi
            // bagi pesan kita selama proses pengiriman
            url = dgram.getAddress();
        }
        if (dgram.getLength() > 0){
            String mesg =
                new String(dgram.getData(), 0, dgram.getLength()));
        }
    }
```

```
}
catch (IOException ioe) {}
catch (Exception e) {}

...


private void sendMesg(String line){
    try {
        byte[] bytes = line.getBytes();
        Datagram dgram = null;

        dgram = dc.newDatagram(bytes, bytes.length, url);

        dc.send(dgram);
    } catch (Exception ioe) {}
}
```

6.7 Latihan

6.7.1 Mendapatkan URL

Buatlah sebuah MIDlet yang mendapatkan HTTP URL. Aplikasi tersebut akan mendapatkan URL dengan method GET dan menampilkan jenis koneksi/ content properties (jika tersedia): Reponse Code, Response Message, Length, Type, Encoding, Expiration dan Last Modified Date.


+5550001 - DefaultColorPhone
MIDlet Help
Java
abc
Exercise 6.1
URL
http://localhost:8084/Chapter07/bong.wav
URL Fetcher Response Code: 200
Response Mesg: OK
Length: 24936
Content Type: audio/x-wav
getEncoding: null
Date: 1126432407000
Expiration: 0
Last Modified: 1126278424000
URL: http://localhost:8084/Chapter07/bong.wav
Request Method: GET
Exit
Fetch
SELECT
CLEAR
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
* .
SHIFT
0
- +
SPACE