CMER

Centre for Mobile Education and Research

Creating Network Connections

Overview

- Fetching Data Using HTTP or TCP Sockets
- Required Network Information
- Types of Gateways
- HTTP Connection Steps
- HTTP Authentication
- HTTPS Connections
- Socket Connections
- Datagram Connections
- WIFI
- USB Send/Receive Examples
- Bluetooth

Fetching Data Using HTTP or TCP Sockets

- BlackBerry Java Applications can use HTTP, HTTPS, and TCP socket protocols to establish connections over the wireless network
- When establishing the connection over the cellular network, a BlackBerry Java Application can use one of two wireless gateways to proxy the connection to the Internet or the corporate intranet.
- The application can be designed to rely on the default gateway that is available to the BlackBerry device user, or could be customize though your code to explicitly select a preferred gateway.

Required Network Information

- The following is the network information that is required in order to create connections
  - Determine the name of the wireless network that the BlackBerry device is registered with
  - Verify that the BlackBerry is in network coverage
  - Explicitly selecting a gateway

Required Network Information (Cont.)

- Determining the name of the wireless network
  - The device must be registered with a wireless network for this method to work.
  - This is done by Invoking RadioInfo.getCurrentNetworkName().
  - Example
    - String networkName = RadioInfo.getCurrentNetworkName();
    - System.out.println (“Network Name: “ + networkName);

Required Network Information (Cont.)

- Verify that the BlackBerry is in network coverage
  - Use the CoverageInfo class and CoverageStatusListener interface of the net.rim.device.api.system package to make sure that the device is in network coverage

Required Network Information (Cont.)

- Explicitly selecting a gateway
  - Set up your application to use the preferred gateway for a connection and to use the default gateway only when the preferred gateway is not available.
  - There are two types of gateways that can be setup
    - Using the BlackBerry Enterprise Server as an intranet gateway
    - Using the wireless service provider's Internet gateway

Types of Gateways

- Using the BlackBerry Enterprise Server (BES) as an intranet gateway
  - All traffic between your application and the BES is automatically encrypted using AES or triple DES encryption.
  - Since the BES resides behind the corporate firewall and provides inherent data encryption, these applications can communicate with application servers and web servers that reside on the corporate intranet.
  - If your application connects to the Internet rather than to the corporate intranet, could possibly use the BES that belongs to the customer as a gateway as well.
    - Network requests travel behind the corporate firewall to the BES, which makes the network request to the Internet through the corporate firewall.
    - An IT policy can be set to enforce that the BESis the gateway for all wireless network traffic, including traffic destined for the Internet.

Types of Gateways (Cont.)

- Using the wireless service provider's Internet gateway
  - applications for the devices can connect to the Internet using the Internet gateway that the wireless service provider provides.
  - Most wireless service providers provide their own Internet gateway that offers direct TCP/IP connectivity to the Internet.
  - Some operators also provide a WAP gateway that lets HTTP connections occur over the WAP protocol. Either of these gateways can be used to establish connections to the Internet.
  - If the application is intended for users on a specific wireless network, this approach is recommended.
  - If the application is intended for a variety of wireless networks, testing your program against the different Internet gateways and achieving a consistent and reliable experience can be challenging.

HTTP Connection Steps

- Make sure that before you open an HTTP connection, verify that the device is in a network coverage area.
  - Use the *CoverageInfo* class and *CoverageStatusListener* interface of the *net.rim.device.api.system* package to make sure that the BlackBerry device is in network coverage.
- Open an HTTP connection
  - Invoke *Connector.open(),* specifying http as the protocol.
  - Cast the returned object as an *HttpConnection* or a *StreamConnection* object.

```
HttpConnection conn = null;
String URL = "http://www.myServer.com/myContent";
conn = (HttpConnection)Connector.open(URL);
```

HTTP Connection Steps (Cont.)

- Set the HTTP request method (GET or POST)
  - Invoke *HttpConnection.setRequestMethod().*

    conn.setRequestMethod(HttpConnection.POST);
- Set header fields
  - Invoke *setRequestProperty()* on the *HttpConnection*.

    conn.setRequestProperty("User-Agent", "BlackBerry/3.2.1");
- Retrieve header fields
  - Invoke *getRequestProperty()* on the *HttpConnection*.

    String lang = conn.getRequestProperty("Content-Language");

HTTP Connection Steps (Cont.)

- **Send and receive data**
  - **Invoke *openInputStream()* and *openOutputStream()* on the *HTTPConnection*.**

```
InputStream in = conn.openInputStream();
OutputStream out = conn.openOutputStream();
```

HTTP Authentication

- Before opening an HTTP connection, verify that the BlackBerry device is in network coverage.
- Open an HTTP connection.
  - Invoke *Connector.open()*, using the HTTP location of the protected resource.
  - Cast and store the returned object as a *StreamConnection*.

```
StreamConnection s =
  (StreamConnection)Connector.open("http://my
  site.com/myProtectedFile.txt");
```

  - Cast and store the *StreamConnection* object as an *HTTPConnection* object.

```
HttpConnection httpConn = (HttpConnection)s;
```

HTTP Authentication (Cont.)

- Determine the status of the HTTP connection.
  - Invoke HttpConnection.getResponseCode().

```
int status = httpConn.getResponseCode();
```

- Retrieve login information from a user.
  - Create code that manages an unauthorized HTTP connection attempt.

```
int status = httpConn.getResponseCode();
switch (status)
case (HttpConnection.HTTP_UNAUTHORIZED);
```

  - Create a run()method and within it implement a dialog object to ask the BlackBerry device user for login information.

```
UiApplication.getUiApplication().invokeAndWait(new Runnable()) {
        public void run()
        {
                dialogResponse = Dialog.ask;
                (Dialog.D_YES_NO,"Unauthorized Access:\n Do you
                   wish to log in?");
        }
}
```

HTTP Authentication (Cont.)

- Process the response of the user.
  - Create code that manages a Yes dialog response.
  - Retrieve the login information and close the current connection.

```
if (dialogResponse == Dialog.YES)
{String login = "username:password";
//Close the connection.
s.close();
```

  - Encode the login information.

```
byte[] encoded =
Base64OutputStream.encode(login.getBytes(), 0,
login.length(), false, false);
```

HTTP Authentication (Cont.)

- Use the login information to access the protected resource.
  - Open a new HTTPConnection and add the authorization header by invoking *HTTPConnection.setRequestProperty()*using the encoded login information.

```
s =
   (StreamConnection)Connector.open("http://mysite.com/
   myProtectedFile.txt ");
httpConn = (HttpConnection)s;
httpConn.setRequestProperty("Authorization", "Basic " +
   new String(encoded));
```

Using HTTPS Connections

- Make sure that before you open an HTTP connection, verify that the device is in a network coverage area.
  - Use the *CoverageInfo* class and *CoverageStatusListener* interface of the *net.rim.device.api.system* package to make sure that the BlackBerry device is in network coverage.
- Open an HTTP connection
  - Invoke *Connector.open(),* specifying HTTPS as the protocol.
  - Cast the returned object as an *HttpsConnection* object.

```
HttpsConnection stream =
  (HttpsConnection)Connector.open("https://host:443/");
```

HTTPS Connections (Cont.)

- **Specify the connection mode.**
  - **If your BlackBerry is associated with a BES and uses an HTTPS proxy server that requires authentication, you will not be able to use end-to-end TLS.**
  - **To open an HTTPS connection in end-to-end mode, add one of the following parameters to the connection string that passes to *Connector.open():***
    - **Specify that an end-to-end HTTPS connection must be used from the device to the target server: EndToEndRequired.**

HTTPS Connections (Cont.)

- Specify that an end-to-end HTTPS connection should be used from the BlackBerry device to the target server. If the device does not support end-to-end TLS, and the BlackBerry device user permits proxy TLS connections, then a proxy connection is used: EndToEndDesired.

```
HttpsConnection stream =
  (HttpsConnection)Connector.open("http
  s://host:443/;EndToEndDesired");
```

Socket Connections

- Although you can implement HTTP over a socket connection, you should use an HTTP connection for the following reasons:
  - Socket connections do not support BlackBerry MDS features, such as push.
  - Applications that use socket connections typically require significantly more bandwidth than BlackBerry Java Applications that use HTTP connections
- When Opening a Socket connection, the *deviceside* parameter needs to specify whether or not the connection uses BlackBerry MDS Services *[(deviceside=false)*] or direct *TCP [(deviceside=true)].*

Socket Connection (Cont.)

- If you do not specify the optional *deviceside* parameter, the following results occur:
  - The connection uses direct TCP by default for any BlackBerry on the iDEN network (Series 6510, 7510, 7520, 7100i).
  - On all other BlackBerry devices, BlackBerry MDS Services is used by default
  - If the MDS Services is not available, the BlackBerry device uses direct TCP.

Creating Socket Connections

- Before opening a socket connection, verify that the device is in network coverage.
- Open a socket connection using the MDS Services.
  - Invoke *Connector.open()*, specifying *socket* as the protocol and appending the deviceside=false parameter to the end of the URL. The application must input their local machine's IP explicitly because localhost is not supported.

```
private static String URL =
   "socket://local_machine_IP:4444;deviceside=false";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

Creating Socket Connections (Cont.)

- Open a socket connection over direct TCP
  - Invoke *Connector.open(),* specifying socket as the protocol, appending the *deviceside=true* parameter to the end of the URL.

```
private static String URL =
  "socket://local_machine_IP:4444;devicesid
  e=true";
StreamConnection conn = null;
conn =
  (StreamConnection)Connector.open(URL);
```

Creating Socket Connections (Cont.)

- Open a socket connection over direct TCP, specifying APN information.
  - Invoke *Connector.open()*, specifying socket as the protocol, appending the *deviceside=true* parameter to the end of the URL. Specify the following APN parameters:
    - The APN parameter contains the APN over which the connection will be made.
    - The tunnelauthusername parameter contains the user name to connect to the APN.
    - The tunnelauthpassword parameter contains the password for the tunnelauthusername.
  - The *tunnelauthusername* and *tunnelauthpassword* parameters can be omitted from the connection URL if they are not required by the APN.
  - If you are creating a direct TCP connection, use these parameters.
  - Connections through the MDS Services are automatically routed by the device; therefore, no APN information is required.

```
private static String URL =
    "socket://local_machine_IP:4444;deviceside=true;apn=internet.com;tu
    nnelauthusername =user165;tunnelauthpassword=user165password";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

Creating Socket Connections (Cont.)

- Send and receive data.
  - Invoke openInputStream() and openOutputStream().

```
OutputStreamWriter _out = new
        OutputStreamWriter(conn.openOutputStream());
String data = "This is a test";
int length = data.length();
_out.write(data, 0, length);
InputStreamReader _in = new
        InputStreamReader(conn.openInputStream());
char[] input = new char[length];
for ( int i = 0; i < length; ++i ) {
   input[i] = (char)_in.read();
};
```

Creating Socket Connections (Cont.)

- Close the Socket connection.
  - Invoke close() on the input and output streams and the socket connection.

```
_in.close();
_out.close();
conn.close();
```

  - Each of the *close()* methods throws an IOException.
  - Make sure that your application implements exception handling.

Datagram Connections

- Datagrams are independent packets of data that applications send over networks.
- A Datagram object is a wrapper for the array of bytes that is the payload of the datagram.
- You would use a datagram connection to send and receive datagrams.
- To use a datagram connection, you must have your own infrastructure to connect to the wireless network, including an APN for GPRS networks.
- Using UDP connections requires that you work closely with service providers and verify that the provider supports UDP connections.

Using Datagram Connections

- Before opening a datagram connection, verify that the device is in network coverage.
- Open a datagram connection
  - Invoke *Connector.open(),* specifying udp as the protocol.
  - Cast the returned object as a *DatagramConnection* object.

    (DatagramConnection)Connector.open("udp://host:dest_port[;src_port]/apn");
    - where:
      - host is the host address in dotted ASCII-decimal format.
      - dest-port is the destination port at the host address (optional for receiving messages).
      - src-port is the local source port (optional).
      - apn is the network APN in string format.

Datagram Connections (Cont.)

- Receive datagrams from all ports at the specified host.
  - Omit the destination port in the connection string.
- Open a datagram connection on a non-GPRS network.
  - Specify the source port number, including the trailing slash mark.
  - Eg. the address for a CDMA network connection would be udp://121.0.0.0:2332;6343/.
  - You can send and receive datagrams on the same port.

Datagram Connections (Cont.)

- Create a datagram.
  - Invoke DatagramConnection.newDatagram().

    Datagram outDatagram = conn.newDatagram(buf, buf.length);
- Add data to a diagram.
  - Invoke Datagram.setData().

    byte[] buf = new byte[256];

    outDatagram.setData(buf, buf.length);

Datagram Connections (Cont.)

- Send data on the datagram connection.
  - Invoke send() on the datagram connection.

    conn.send(outDatagram);
  - If an application attempts to send a datagram on a datagram connection and the recipient is not listening on the specified source port, an IOException is thrown.
  - Make sure that the BlackBerry Java Application implements exception handling.

Datagram Connections (Cont.)

- Receive data on the datagram connection.
  - Invoke *receive()* on the datagram connection. Since the *receive()* method blocks other operations until it receives a data packet, use a timer to retransmit the request or close the connection if a reply does not arrive.

```
byte[] buf = new byte[256];
Datagram inDatagram = conn.newDatagram(buf,
    buf.length);
conn.receive(inDatagram);
```

Datagram Connections (Cont.)

- Extract data from a datagram.
  - Invoke *getData().* If you know the type of data that you are receiving, convert the data to the appropriate format.

    String received = new String(inDatagram.getData());
- Close the datagram connection.
  - Invoke *close()* on the input and output streams, and on the datagram connection object.

Wi-Fi

- Determine if the transceiver for the WLAN is on:
  - Create an IF statement that tests the value of RadioInfo.WAF_WLAN and the value returned by RadioInfo.getActiveWAFs()

  ```
  if ( ( RadioInfo.getActiveWAFs() & RadioInfo.WAF_WLAN ) != 0 ) { … }
  ```
- Determine if the transceiver is connected to an access point:
  - From the net.rim.device.api.system package, import the WLANInfo class
  - Create an IF statement that tests the value of WLANInfo.WLAN_STATE_CONNECTED and the value returned by WLANInfo.getWLANState()

  ```
  if (WLANInfo.getWLANState() == WLANInfo.WLAN_STATE_CONNECTED) {…}
  ```
  - The WLANInfo.getWLANState() method checks if a BlackBerry device has an IP address and can transfer data over a Wi-Fi network. If the transceiver for the WLAN wireless access family is off, this method returns WLANInfo.WLAN_STATE_DISCONNECTED

Wi-Fi (Cont.)

- You can retrieve status information such as the data rate of the connection, the wireless LAN standards used (802.11a, b or g), the SSID of the associated access point, or the name of the Wi-Fi profile in use
- The transceiver for the WLAN wireless access family must be connected to a wireless access point.
- From the net.rim.device.api.system package, import the WLANInfo class.
- Invoke WLANInfo.getAPInfo(), storing a reference to WLANInfo.WLANAPInfo that this method returns. The WLANInfo.WLANAPInfo object contains a snapshot of the current wireless network.
- WLANInfo.WLANAPInfo info = WLANInfo.getAPInfo();
- If the BlackBerry device is not connected to an access point, the WLANInfo.getAPInfo() method returns null.
- See the API reference for the BlackBerry Java Development Environment for more information about WLANInfo.WLANAPInfo.

Wi-Fi (Cont.)

- Determine if the BlackBerry device is accessing a wireless network through a wireless access point:
  - Invoke the RadioInfo.getNetworkService method using the RadioInfo.WAF_3GPP parameter
  - In the bitmask of the RadioInfo.NETWORK_SERVICE_* flags that the getNetworkService(int)method returns, check to see if the RadioInfo.NETWORK_SERVICE_GAN flag is set in the return value
- When a 3GPP wireless access family generates a transceiver event, determine if the BlackBerry device is accessing a wireless network through a wireless access point
  - When the listener's RadioStatusListener.networkServiceChange(int networked, int service) method is invoked, check for the RadioInfo.NETWORK_SERVICE_GAN flag in the service parameter
  - If this flag is set in the service parameter, the BlackBerry device is accessing a wireless network through a wireless access point

Wi-Fi (Cont.)

- Receive notifications of changes in the connectivity state of a Blackberry device:
  - Use the addListener()methods of the CoverageInfo class
- Determine if the BlackBerry device has enough wireless coverage to attempt a direct TCP connection through a wireless access point:
  - Invoke isCoverageSufficient(COVERAGE_CARRIER,RadioInfo.WAF_WLAN,false)
- Determine if the BlackBerry device has enough wireless coverage to attempt a WLAN enterprise connection through a wireless access point:
  - Invoke isCoverageSufficient(COVERAGE_MDS, RadioInfo.WAF_WLAN, false)

Wi-Fi (Cont.)

- Open a Wi-Fi socket connection:
  - Invoke Connector.open(), specify socket as the protocol, and append the deviceside=true parameter and the interface=wifi parameter to the end of the URL string value

```
private static String URL = "socket://local_machine_IP:4444;
deviceside=true;interface=wifi";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

- Open a Wi-Fi HTTP connection:
  - Invoke Connector.open(), specify http as the protocol, and append the interface=wifi parameter to the end of the URL string value
  - Cast the returned object as an HttpConnection or a StreamConnection object

```
HttpConnection conn = null;
String URL =
    "http://www.myServer.com/myContent;deviceside=true;interface=
    wifi";
conn = (HttpConnection)Connector.open(URL);
```

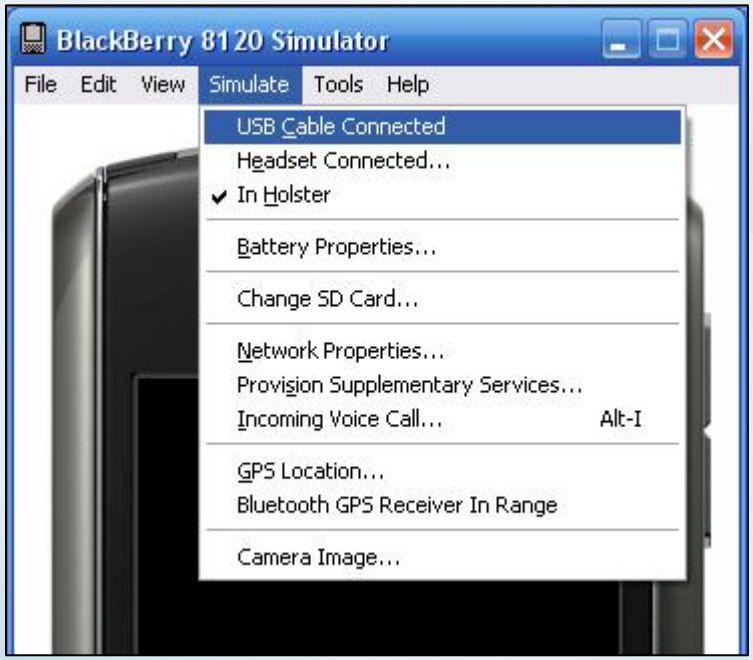
Wi-Fi (Cont.)

- Open a Wi-Fi HTTPS connection:
  - Invoke Connector.open(), specify https as the protocol, and append the interface-wifi parameter to the end of the URL string value
  - Cast the returned object as an HttpsConnection object

```
HttpsConnection conn = null;
String URL = "https://host:443/;
deviceside=true;interface=wifi";
conn = (HttpsConnection)Connector.open(URL);
```

USB/Serial

- USB and serial connections allow BlackBerry applications to communicate with desktop applications and peripheral devices connected to the BlackBerry
- It is possible to simulate a USB connection using the BlackBerry Simulator



USB Send Example

USB Send

```
//create the comm connection with USB as the port
StreamConnection con = _
    (StreamConnection)Connector.open("comm:COM1;baudrate=9600;bitsperchar
    =8 ;parity=none;stopbits=1");

//create a data output stream from the USB connection stream
DataOutputStream dos = con.openDataOutputStream();

//the string to send
String sdata = "This is a test";

//send the data
dos.writeChars(sdata);

//close the connections
dos.close();
con.close();
```

USB Send Example (Cont.)

- In the example, a StreamConnection object is created through the Connector.open() method
- This object represents the USB connection
- In the example this connection is connected on com port 1, with baud rate at 9600bps, 8 bits per character, no parity, and 1 stop bit
- Next, a DataOutputStream object is created to be used for transmission
- A test message, “this is a test”, is then created and assigned to the String sdata
- The test message is then put into the output stream
- The connection and outputstream are closed

USB Receive Example

USB Receive

```
//create the comm with USB as the port
StreamConnection con = _
    (StreamConnection)Connector.open("comm:COM1;baudrate=960
    0;bitsperchar=8 ;parity=none;stopbits=1");

//create a data input stream from the USB connection stream
DataInputStream dis = con.openDataInputStream();

//receive the data
String rdata = dis.readUTF();

//close the connections
dis.close();
con.close();
```

USB Receive Explained

- In the example, a StreamConnection object is created through the Connector.open() method
- This object represents the USB connection
- In the example this connection is connected on com port 1, with baud rate at 9600bps, 8 bits per character, no parity, and 1 stop bit
- Next, a DataInputStream object is created to be used for transmission
- The String sdata is used to store the incoming data
- The connection and inputstream are closed

Bluetooth

- BlackBerry 7100, 7250, 7290, and 7520 were the first to support Bluetooth, version 1.1. All later BlackBerry devices with Bluetooth wireless technology use version 2.0
- Applications are able to create Bluetooth connections using the Bluetooth Serial Port Profile on any Bluetooth enabled BlackBerry device
- Bluetooth Serial Port Profile, part of the JSR 82 implementation may be used to initiate a server or client Bluetooth serial port connection to a computer or other Bluetooth enabled devices
- The JSR 82 implementation added some additional Bluetooth wireless technology profiles that can be used by third-party applications.
  - Object Push Profile (OPP)
  - Object Exchange (OBEX)

Bluetooth

- Bluetooth API - net.rim.device.api.bluetooth
  - BluetoothSerialPortListener
  - BluetoothSerialPort
  - BluetoothSerialPortInfo
- Unlike USB connections, Bluetooth connections are not possible to simulate
- Bluetooth development kits for the BlackBerry simulation environment such as Casira available from Cambridge Silicon Radio (CSR)
- More on Casira at http://www.btdesigner.com/devcasira.htm