

Java 2 Micro Edition Sockets and SMS



F. Ricci
2010/2011

Content

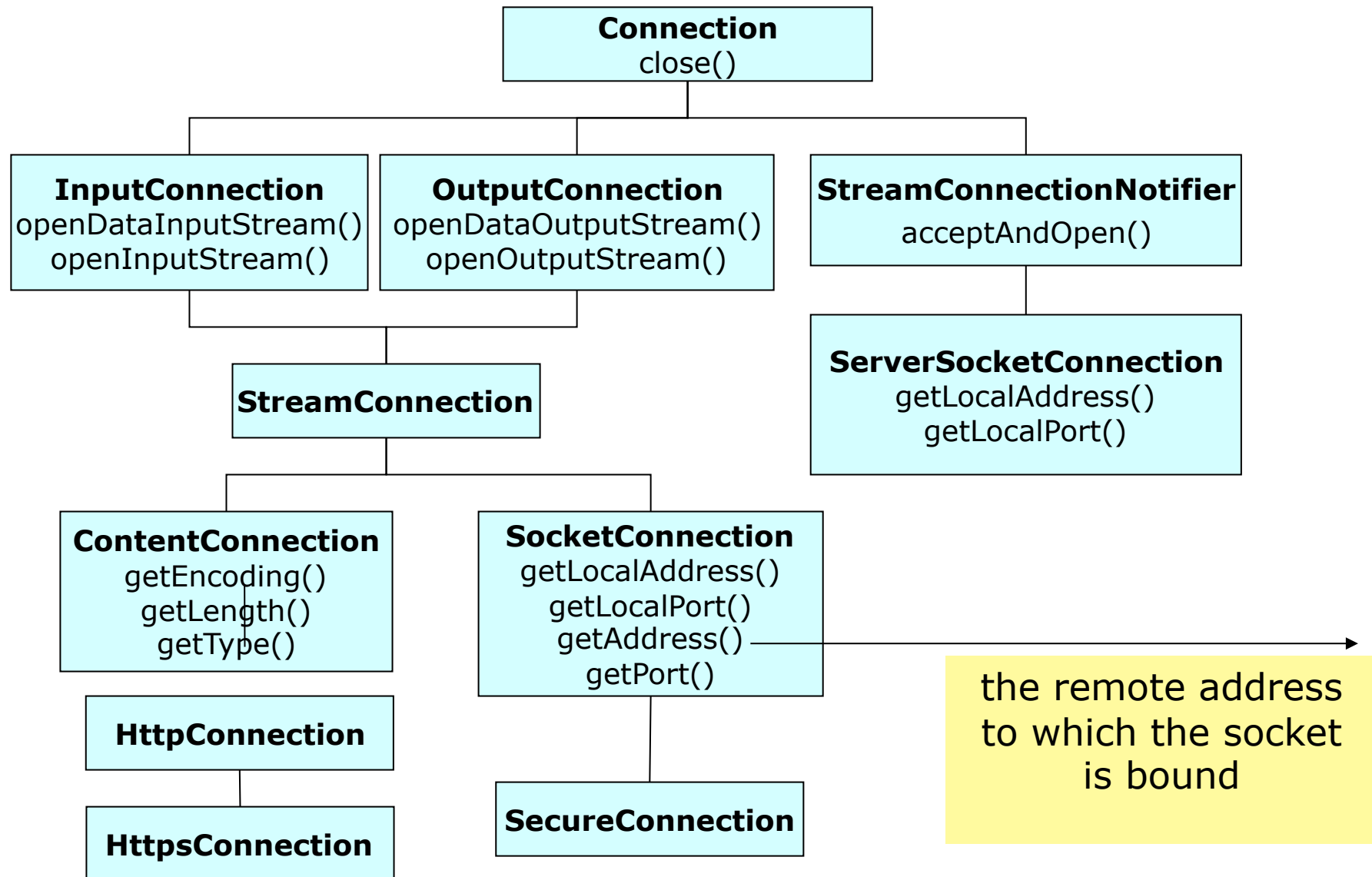
- Other Connection Types
- Responding to Incoming Connections
 - Socket and Server Socket
- Security
 - Permissions
 - Security domains
 - Midlet signing
- Wireless Messaging
- Responding to incoming messages
- Lightweight UI Toolkit (LWUIT)

Other Connection Types

- ❑ MIDP specification **requires** only HTTP and HTTPS connections, but devices may also choose to implement access to serial ports through the generic connection framework
- ❑ Additional **connection types**, their supporting connection interface and example connection string:

Type	Interface	Example
Socket	SocketConnection	socket://localhost:79
Server socket	ServerSocketConnection	socket://:129
TLS or SSL socket	SecureConnection	ssl://localhost:79
Serial port	CommConnection	comm:com0;baudrate=19200

Socket and ServerSocket



Socket and ServerSocket

- ❑ In a **socket** connection: a socket is accessed using a generic connection string with an explicit **host** and **port** number
 - E.g.: `socket://host.com:79`
 - The MIDlet **connects to a server's socket**
- ❑ `ServerSocketConnection` provides the ability to **listen** for incoming socket connections while a MIDlet is running
- ❑ A **server socket** is accessed using a generic connection string with the host **omitted**
 - E.g.: `socket://:79` defines an **inbound** server socket on port 79
 - The MIDlet is the server to whom other components will connect
- ❑ The `acceptAndOpen()` method of `ServerSocket` returns a `StreamConnection` instance (who called the server).

Responding to Incoming Connections (I)

- ❑ MIDP allow MIDlets to be launched in response to incoming network connections (**socket** or even **SMS**)
- ❑ The name of this technique is ***push***
- ❑ A MIDlet may **register** for push connection in two ways
 - At **runtime** calling static methods of `PushRegistry`
 - At **installation time** using special entry in the **application descriptor** (JAD)
- ❑ After having registered:
 - Inside the MIDlet, catch the incoming connection using the `acceptAndOpen()` method of `ServerSocket` (previously open with `Connector.open()`).

Responding to Incoming Connections (II)

- For example: a **web server** in MIDlet called `PatchyMidlet`
- This MIDlet responds to incoming socket connections on port 82 (or whatever you like, e.g., 80)
 - If there are problems use 8200!
- If you want to register at runtime you have to write in your midlet some code like this

```
PushRegistry.registerConnection("socket://:8200,  
PatchyMIDlet, "*");
```

- The first parameter indicates the listening socket, the second the MIDlet to run, and the third a filter on incoming IP address - the * indicates that all addresses are accepted
- To register the connection at installation time, simply put the following line in the descriptor (JAD)

```
MIDlet-Push-1: socket://:8200, PatchyMidlet, *
```

PatchyMIDlet.java

```
// import omitted

public class PatchyMIDlet extends MIDlet implements CommandListener,
    Runnable {
    private Display mDisplay;
    private Form mForm;

    private ServerSocketConnection mServerSocketConnection;
    private boolean mTrucking = true;

    public void startApp() {
        mDisplay = Display.getDisplay(this);

        if (mForm == null) {
            mForm = new Form("PatchyMIDlet");
            mForm.addCommand(new Command("Exit", Command.EXIT, 0));
            mForm.setCommandListener(this);
        }

        Thread t = new Thread(this);
        t.start();

        mDisplay.setCurrent(mForm);
    }
}
```

[code](#)

PatchyMIDlet.java

```
public void pauseApp() {}
public void destroyApp(boolean unconditional) { shutdown(); }
private void log(String text) { log(null, text); }
private void log(String label, String text) {
    StringItem si = new StringItem(label, text);
    si.setLayout(Item.LAYOUT_NEWLINE_AFTER);
    mForm.append(si);
}

private void shutdown() {
    mTrucking = false;
    try { mServerSocketConnection.close(); }
    catch (IOException ioe) {}
}

public void commandAction(Command c, Displayable s) {
    if (c.getCommandType() == Command.EXIT) {
        shutdown();
        notifyDestroyed();
    }
}
```

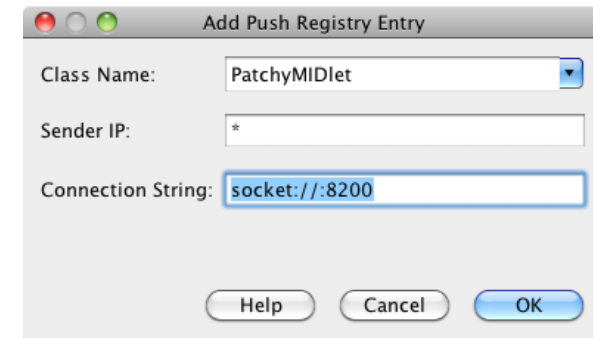
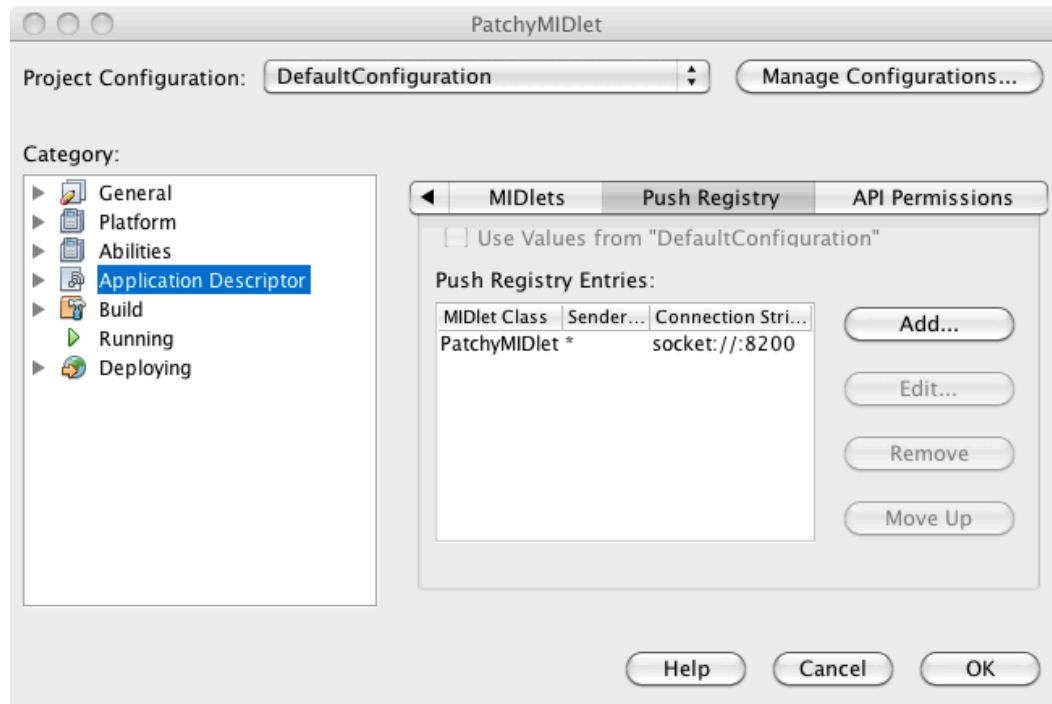
PatchyMIDlet.java

```
public void run() {
    try {
        mServerSocketConnection = (ServerSocketConnection)
            Connector.open("socket://:82");
        log("Startup complete.");
        SocketConnection sc = null;
        while (mTrucking) {
            sc = (SocketConnection)
                mServerSocketConnection.acceptAndOpen();
            log("client: ", sc.getAddress());
            Reader in = new InputStreamReader(
                sc.openInputStream());
            String line;
            while ((line = readLine(in)) != null) ;
            // Ignoring the request, send a response.
            PrintStream out = new PrintStream(sc.openOutputStream());
            out.print("HTTP/1.1 200 OK\r\n\r\n");
            out.print(getMessage());
            out.close();
            in.close();
            sc.close();
        }
    }
    catch (Exception e) {log("exception: ", e.toString());}}
```

PatchyMIDlet.java

```
private String readLine(Reader in) throws IOException {
    StringBuffer line = new StringBuffer();
    int i;
    while ((i = in.read()) != -1) {
        char c = (char)i;
        if (c == '\n') break;
        if (c == '\r') ;
        else line.append(c);
    }
    if (line.length() == 0) return null;
    return line.toString();
}
private java.util.Random mRandom = new java.util.Random();
private String getMessage() {
    int i = Math.abs(mRandom.nextInt()) % 5;
    String s = null;
    switch (i) {
        case 0: s = "Above all the others we'll fly"; break;
        case 1: s = "There is no reason to hide"; break;
        case 2: s = "I dreamed about Ray Charles last night"; break;
        case 3: s = "Someone keeps moving my chair"; break;
        case 4: s = "Joseph's face was black as night"; break;
        default: break;
    }
    return s;}}
```

Responding to Incoming Connections (III)

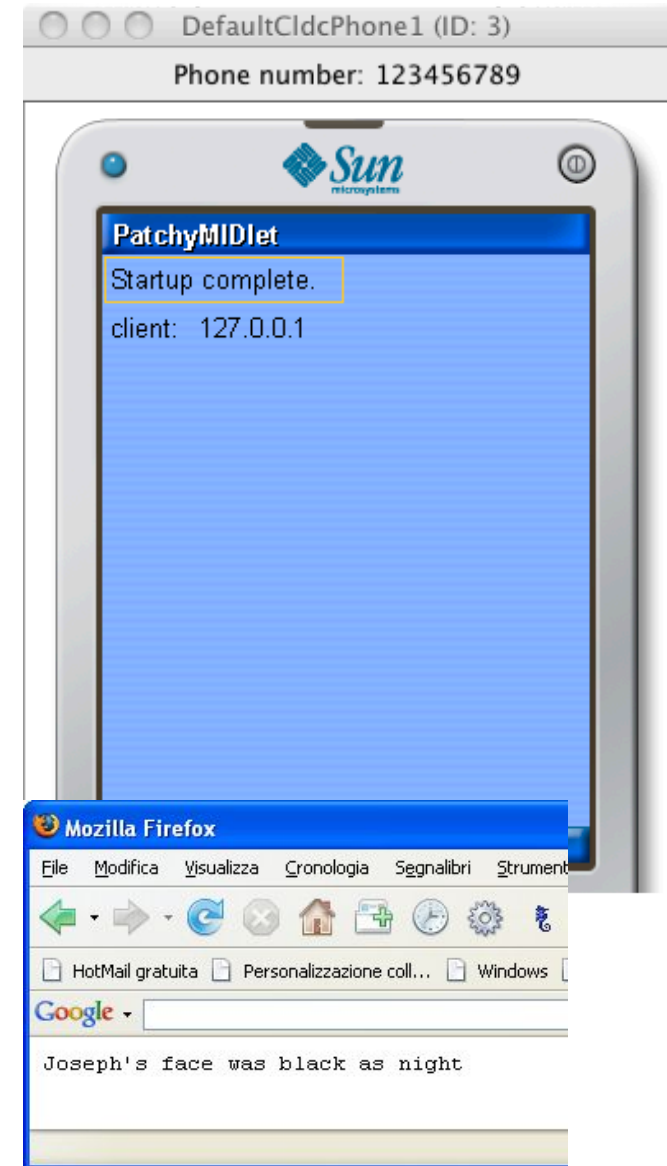


b)

- ❑ The Java ME SDK (or Netbeans) allows you to register and test push connections
- ❑ Click the Project ➤ Application Descriptor
- ❑ And then choose the Push Registry, and “add” what is shown in figure b)

Responding to Incoming Connections (IV)

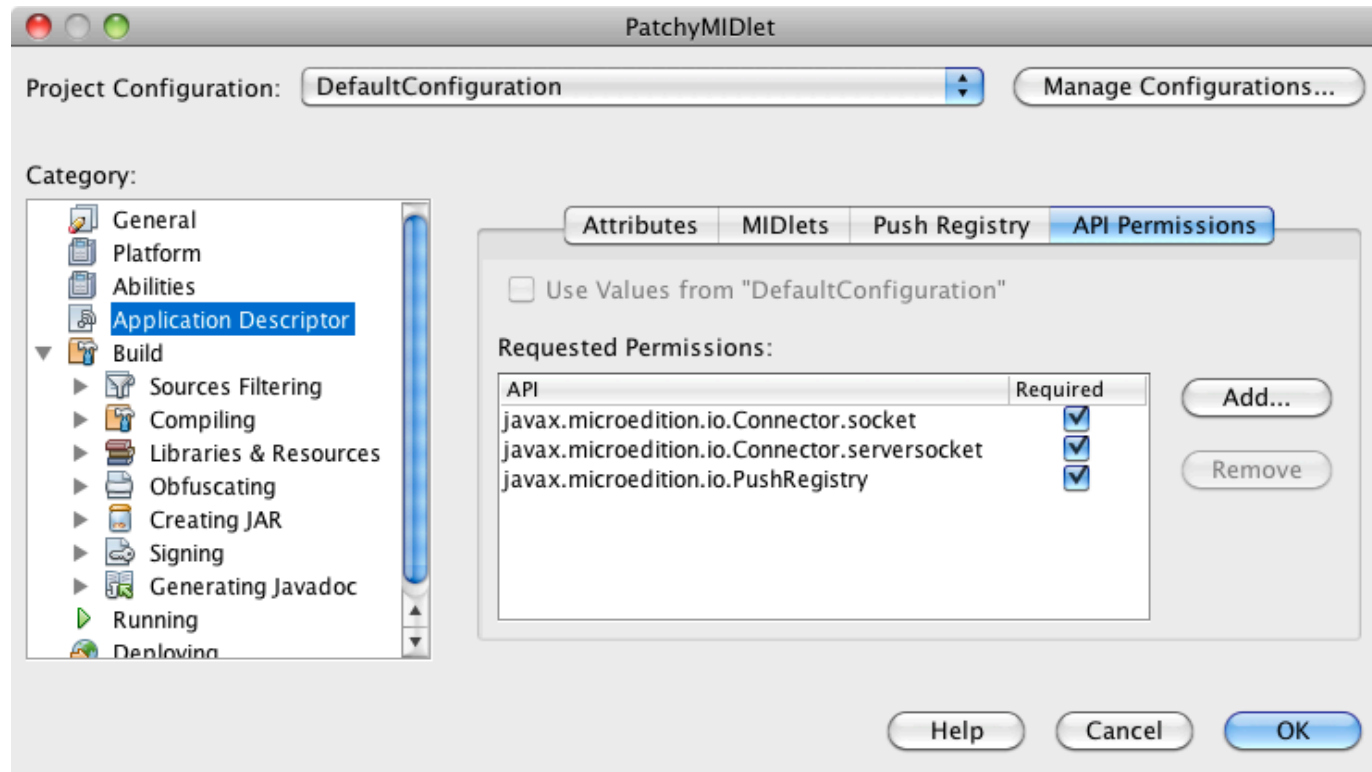
- ❑ To test the push notification, you'll have to package the application and then deploy it on the SDK (or NetBeans) via OTA
- ❑ Choose Project ► **Run via OTA**
- ❑ You'll see emulator pop up showing its Application Management Software (AMS)
- ❑ You'll see other prompts during the installation, say yes to everything
- ❑ The emulator is now running, listening for incoming connections, even though no MIDlets are running
- ❑ Call it at <http://localhost:8200>



Permissions and Security Domains

- ❑ MIDP 2.0 includes a **security framework** that is designed to prevent MIDlets from running up your phone bill by making unauthorized network connections
- ❑ There are:
 - **Permissions** and
 - **Security Domains**
- ❑ **Permission** have names corresponding to the API that they protect and MIDlet suites can indicate their need for certain kinds of permissions through attributes in the MIDlet suite descriptor
- ❑ MIDlets **must have permission** to perform sensitive operations, such as connecting to the network.

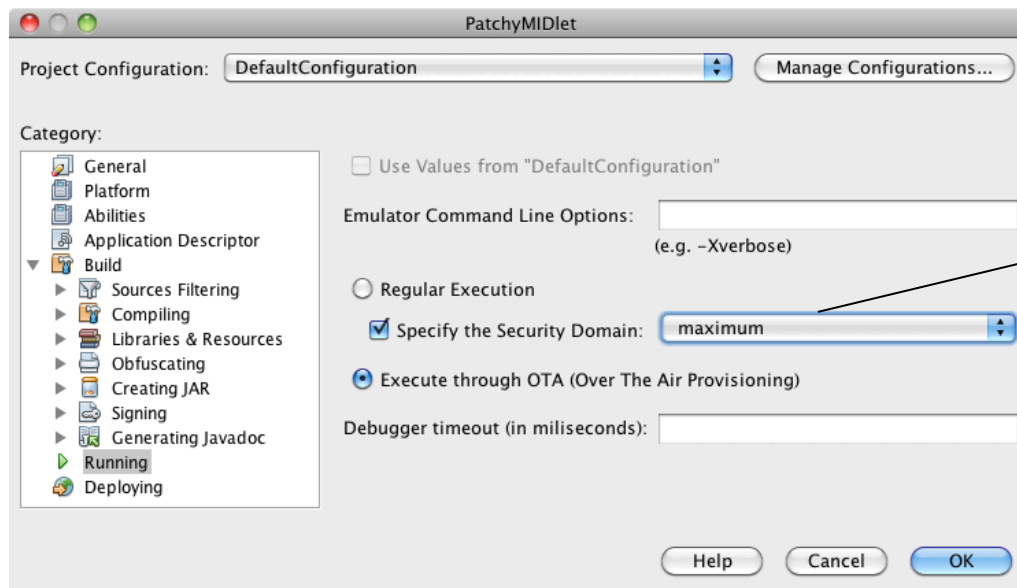
Setting Permissions in NetBeans and WTK3.0



- ❑ This permissions must be declared if you want to sign your MIDlet
- ❑ Access the "project properties" and then the "application description"
- ❑ Permissions are written in the .jad file

Security Policies

- ❑ The Java ME SDK supports the **security policies** defined by both JSR 185 (Java Technology for the Wireless Industry or **JTWI**) and JSR 248 (Mobile Service Architecture or **MSA**)
- ❑ When you run a MIDlet in the toolkit it runs in the `unidentified_third_party` **security domain** by default
- ❑ To change this you must go to "project properties" and then the "running"



Set to maximum domain

Protection Domains

- ❑ The **unidentified_third_party** domain provides a high level of security for applications whose origins and authenticity cannot be determined - the user is prompted frequently when the application attempts a sensitive operation
- ❑ The **identified_third_party domain** is for MIDlets whose origins have been determined using cryptographic certificates - permissions are not granted automatically but the user will be prompted less
- ❑ The **manufacturer** domain is intended for MIDlet suites whose credentials originate from the manufacturer's root certificate (e.g. Nokia)
- ❑ MIDlets in the **minimum** domain are **denied all permissions**
- ❑ MIDlets in the **maximum** domain are **granted all permissions.**

Protection Domains and OTA

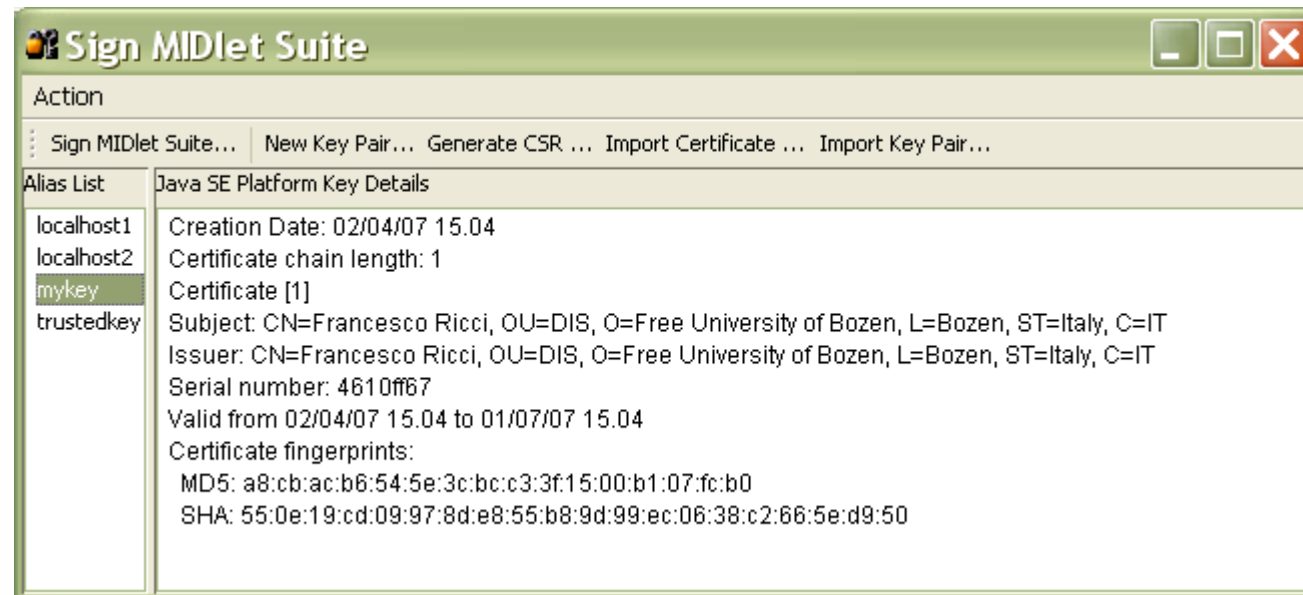
- ❑ Your packaged MIDlet suite is installed directly into the emulator - it is placed in a protection domain at **installation time**
- ❑ The emulator uses **public key cryptography** to determine the protection domain of installed MIDlet suites
- ❑ *If the MIDlet suite is **not signed**, it is placed in the `unidentified_third_party` domain*
- ❑ *If the MIDlet is **signed**, it is placed in whatever protection domain is associated with the root certificate of the signing key's certificate chain*
- ❑ If for instance a company sign a midlet using a key pairs from Verisign, then the midlet is put in the protection domain associated with the root certificate of Verisign, and hence (probably) **identified_third_party domain**.

Signing a Midlet

- ❑ The general process to create a cryptographically signed MIDlet suite is as follows:
 1. The MIDlet author buys a signing key pair from a certificate authority (the CA)
 2. The author signs the MIDlet suite with the signing key pair and distributes their certificate with the MIDlet suite
 3. When the MIDlet suite is installed on the emulator or on a device, the implementation verifies the author's certificate using its own copy of the CA's root certificate - then it uses the author's certificate to verify the signature on the MIDlet suite
 4. After verification, the device or emulator installs the MIDlet suite into the security domain that is associated with the CA's root certificate.

Signing a MIDlet (WTK 2.5.2)

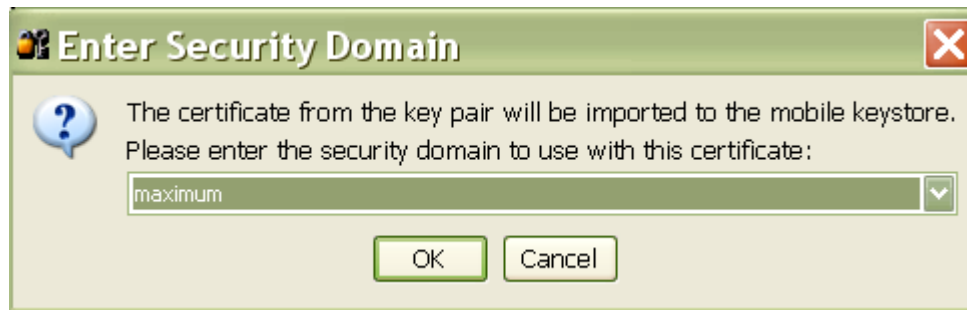
- ❑ To sign a MIDlet suite, **you must declare the required permissions and then package it (do not package it after signing)**
- ❑ Then choose **Project > Sign** from the KToolbar menu



- ❑ select the key you want to use in the **Alias List** and click on the **Sign MIDlet Suite...** button

Creating a new key pair

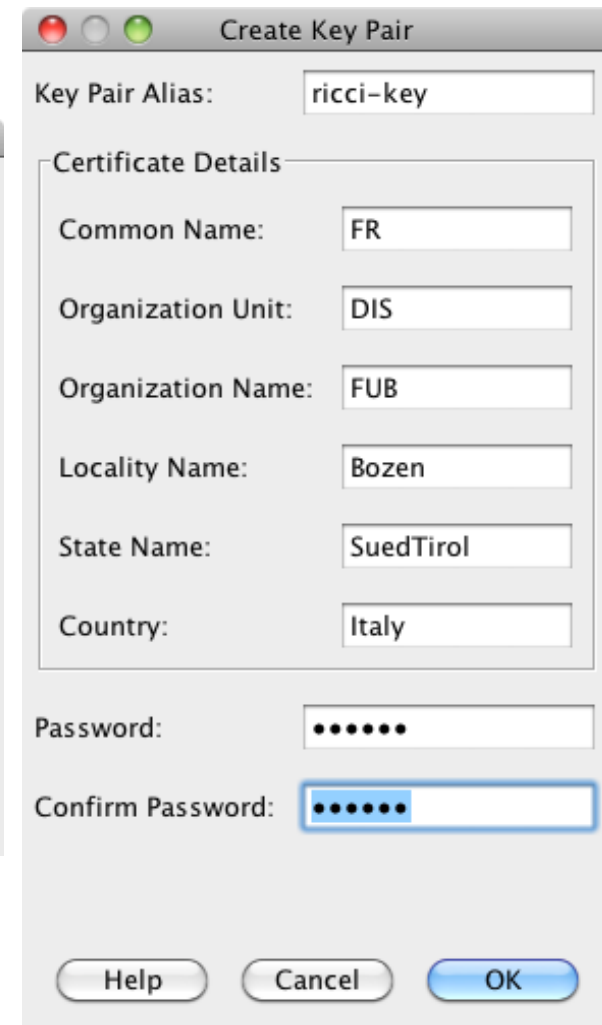
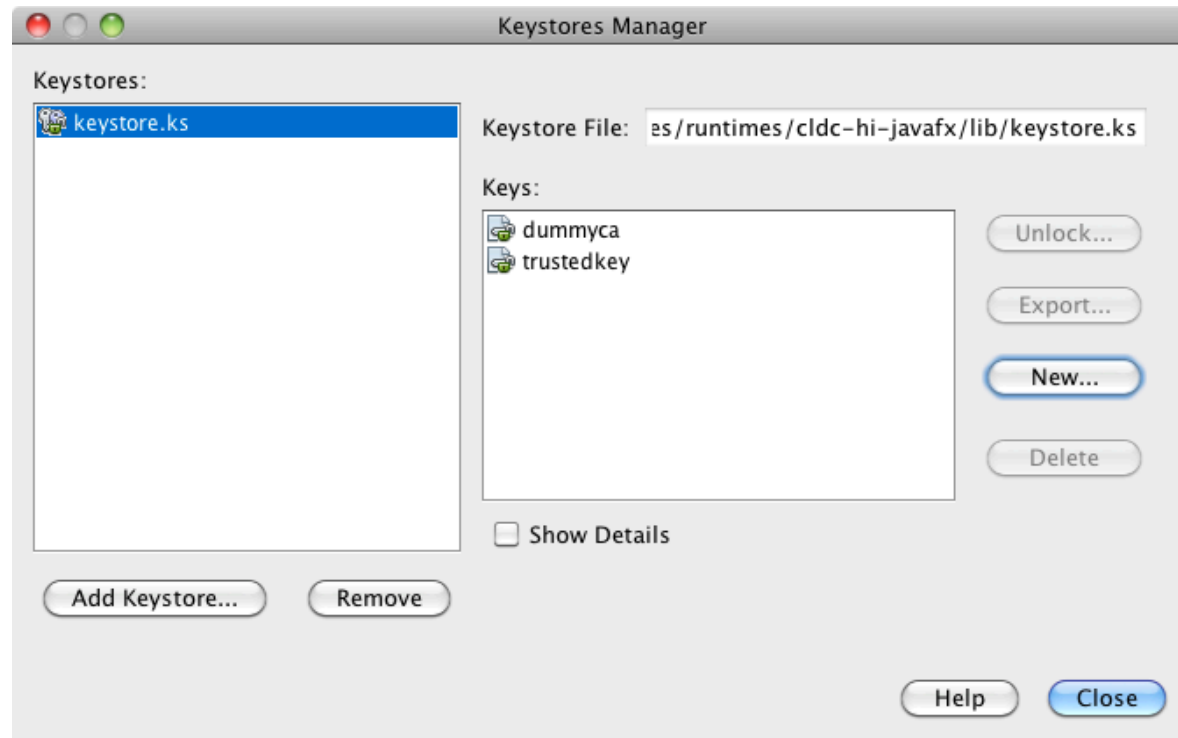
- To create a new key pair **Project > Sign** from the KToolbar menu



- After you click on **Create**, the toolkit prompts you to choose a protection domain where the MIDlet signed with that will be put.

Signing in WTK 3.0 - I

- Create a new key



Signing in WTK 3.0 - II

- Export the key to the emulator (does not work!)

Keystore File: `ts/Resources/runtimes/cldc-hi-javafx/lib/keystore.ks`

Key Pair Alias: `ricci-key`

Certificate Details

ricci-key
Subject: CN=FR, OU=DIS, O=FUB, L=Bozen, ST=SuedTirol, C=Italy
Issuer: CN=FR, OU=DIS, O=FUB, L=Bozen, ST=SuedTirol, C=Italy
Valid: Nov 5, 2010 - May 4, 2011

Emulator: `CLDC Java(TM) Platform Micr...`

Security Domain: `maximum`

Keys Registered in the Emulator:

- Key: 1**
Owner: C=US;O=RSA Data Security, Inc.;OU=Secure Server
Valid from Wed Nov 09 01:00:00 CET 1994 to Fri Jan 08
Security Domain: identified
Enabled: true
- Key: 2**
Owner: ST=state;L=city;O=org;OU=orgUnit;CN=cName

Buttons: Help, Export, Close

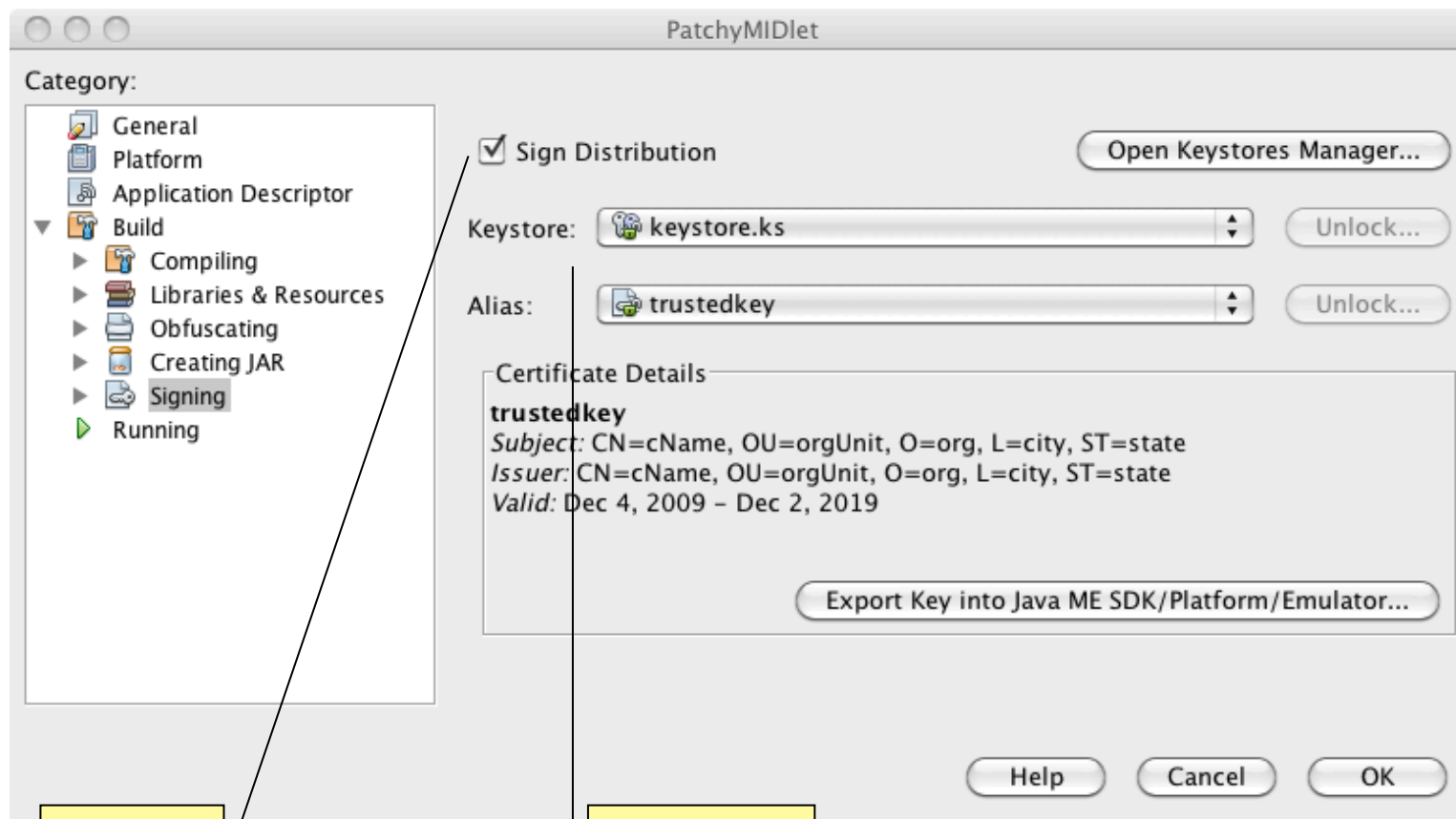
Buttons: Unlock..., Export..., New..., Delete, Close

Select the emulator

... and the security domain ... in a real phone this is determined by the CA certificate.

Running a signed Midlet in WTK 3.0 III

- ❑ In the project properties select: signing
- ❑ Then run the application via OTA



Mark this

Select the key

Installing via OTA a signed MIDlet

- When you install a signed MIDlet something is different



- Then everything runs as before (because the midlet is put in the same domain as before `unidentified_third_party`)
- In the reality you must sign your midlet with a key certified by a certified authority whose certificate is present in the device.

Wireless Messaging API - WMA

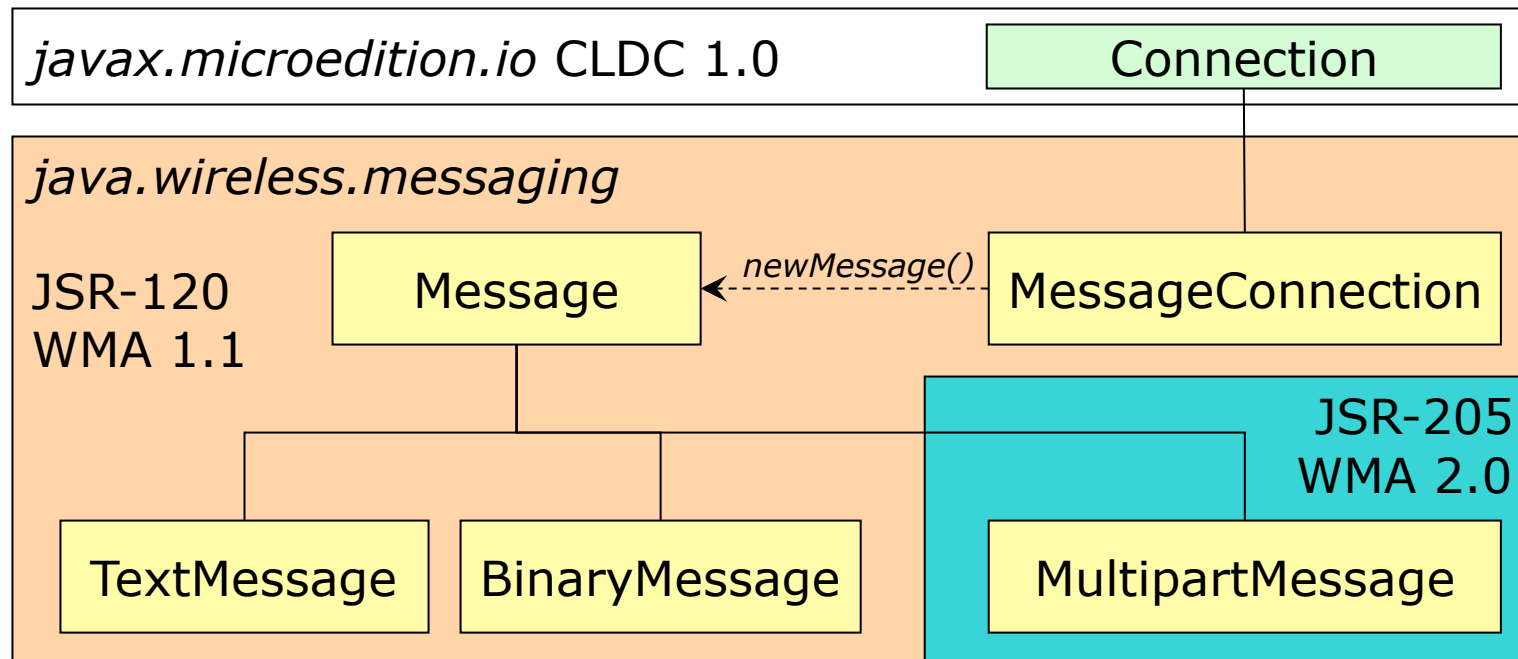
- ❑ WMA 1.1. is an **optional API** that enables MIDP application to leverage the utility of SMS
- ❑ The WMA 2.0 adds support for **MMS** (Multimedia) messages
- ❑ Messages can now be sent and received between phones **without:**
 - Having internet access (additional cost service)
 - Going through an intermediary server (potentially from the carrier at additional cost)
 - Being restricted in terms of routing by the carrier's network
- ❑ The possible applications are unlimited: chat-type applications, interactive gaming, event reminders, e-mail notification, mobile access to corporate resources, informational services, ...

WMA and SMS

- ❑ SMS can be used **with the push registry** to launch MIDlets on your cell phone
- ❑ Using WMA the message can now be **larger in size than a single message** and the content is no longer restricted to simple text message
- ❑ Under the hood, the WMA API will do the following
 - Encode binary message and transmit it through SMS
 - Cut up a long message into segments, and send it via multiple (up to three) SMS messages
- ❑ Two current and finalized JSRs are relevant to WMA
 - **The JSR 120** (WMA 1.1) all devices that support WMA have WMA 1.1 implementation
 - **The JSR 205** (WMA 2.0 support for **multimedia** messaging - MMS) these messages are perfect for carrying images, sound, video, or multimedia presentations.

WMA API

- ❑ WMA is built on top of CLDC and requires only the Generic Connection Framework
- ❑ Like `SocketConnection`, you get a `MessageConnection` by passing in an address to the `open()` method
- ❑ Unlike other connections, **you cannot open in/out stream from it** - `MessageConnections` are only used to send and receive messages



Creating New Messages

- First open a `MessageConnection` like:

```
MessageConnection msgConn = (MessageConnection)
    Connector.open("sms://5550001:1234");
```

- This will create a **client** mode connection to **send messages**

- **server** mode connection - **send and receive SMS:**

```
MessageConnection msgConn = (MessageConnection)
    Connector.open("sms://:1234");
```

- If you want to send messages you first need to create an empty one calling one of the factory methods (of `MessageConnection`):

1. `Message newMessage(String messageType);`
2. `Message newMessage(String messageType, String address);`

- *When you call 1 and when 2?*

- The second method is used when the message is created from a **server mode** connection – determines the address of the receiver.

Sending Text SMS Messages

- There are **three types of messages**:
 - `MessageConnection.TEXT_MESSAGE`
 - `MessageConnection.BINARY_MESSAGE`
 - `MessageConnection.MULTIPART_MESSAGE` (only for WMA 2.0 – to support multimedia)
- **Create an empty message** specifying `MessageConnection.TEXT_MESSAGE` as its type
- Next **set the payload** with the text string that you want to send
- Finally use the `send()` method for **sending**

```
MessageConnection conn = (MessageConnection)
    Connector.open("sms://5550001:1234");

TextMessage txtmessage = (TextMessage)conn.newMessage(
    MessageConnection.TEXT_MESSAGE);

txtmessage.setPayloadText(msgString);

conn.send(txtmessage);
```

Sending Binary SMS Messages

- ❑ First create an **empty message** using the `newMessage()` method of the opened `MessageConnection` specifying `MessageConnection.BINARY_MESSAGE` for its type
- ❑ **Get binary data** into a byte array
- ❑ **Set the payload** of the `BinaryMessage` using

```
Public void setPayloadData(byte[] data);
```
- ❑ Then you can send the message using this `MessageConnection` method

```
void send(Message msg);
```
- ❑ As with any network operations, you should call `send()` on a separate non-GUI thread to avoid hanging the GUI
- ❑ When you are using server mode you must set the recipient address (`message.setAddress()`).

SMS size limitations

- ❑ **Binary** messages are limited to **133 bytes** per SMS
 - Longer messages will be separated by WMA into segments
- ❑ **Text** messages are limited to **66 characters**
 - Longer messages will be separated by WMA into segments
- ❑ `int MessageConnection.numberOfSegments`
(`Message msg`) returns the number of segments required in the underlying protocol
- ❑ **But it is better not to use more than 3 segments,** i.e., 189 characters (text) or 381 bytes (binary), because implementations **MUST** support only 3 segments.

Receiving SMS Messages

- To receive messages using a `MessageConnection`, you have two choices:
 1. Use a **blocking** `receive()` method
 2. **Implement a listener callback** to be notified when a message arrives
- When using the blocking `receive()` method, you typically will be managing your own threads
- Using a listener allows you to code the logic in a callback method, without creating and managing additional threads.

Calling the Blocking `receive()` Method

- ❑ The blocking `receive()` method is on the `MessageConnection` interface
- ❑ **This method will block** the incoming call until an incoming message is available or until the connection is closed
- ❑ Since `receive()` is a blocking call, it should always be called on its own thread
- ❑ A common approach is to create a “receiver thread” to receive messages in an application
- ❑ Closing a `MessageConnection` is a way to release a receiver thread that may be blocked waiting for incoming messages.

Message receive code example

```
conn = (MessageConnection) Connector.open
    ("sms://:1234"); //server mode
msg = conn.receive(); // Blocking here
mSenderId = msg.getAddress();
if (msg instanceof TextMessage) {
    String msgReceived = ((TextMessage)
        msg).getPayloadText();
    // do something with the message here
} else if (msg instanceof BinaryMessage) {
    byte [] msgReceived = ((BinaryMessage)
        msg).getPayloadData();
    // do something with the binary message here
}
```

For Text
Messages

For Binary
Messages

A Nonblocking Approach to Receiving SMS Messages

- `MessageConnection` supports a nonblocking, **event listener-based** way for receiving **SMS** messages - to use this, you will need the following:
 - **Register** a `MessageListener` with the `MessageConnection` - using `setMessageListener()` - the object you registered must implement the `MessageListener` interface (e.g. the midlet itself)
 - **Handle the callback** on the `notifyIncomingMessage(MessageConnection conn)` method of the `MessageListener` interface
- The callback is performed by WMA on a system thread, and the `notifyIncomingMessage()` method must return as soon as possible
- This means that any work should be performed by another thread
- A detailed example is provided in the API description [wma_2.0-fr_wma_ext-102-fr-spec.pdf](http://www.sipac.com/wma_2.0-fr_wma_ext-102-fr-spec.pdf)

Examining Message Headers

- ❑ In addition to the binary or text payload, other interesting information appears in the message header
- ❑ You can access the address information using the following methods on the `Message` interface
 - `public String getAddress();`
 - `public void setAddress();`
- ❑ The `Message` interface is the super interface of both `TextMessage` and `BinaryMessage`
- ❑ The `getAddress()` returns:
 - the **recipient's** address if it is an **outgoing** message, or
 - the **sender** address if it is an **incoming** message
 - *in fact, incoming messages can be reused for replies by just setting a new payload*
- ❑ Another method on the `Message` interface provides access to the timestamp on the message:
 - `public Date getTimestamp();`

Working With SMS APIs - SMSMidlet

- ❑ The `SMSMidlet` example can be used to receive or send SMS messages
- ❑ If you send a message with the word "red" or "blue", an image with a red or blue background will be displayed by the receiving MIDlet
- ❑ You can enter the address that you want to send SMS messages to into the "Connect to:" text field
- ❑ Using the menu you can then send either a "red" or "blue" SMS message to the recipient
- ❑ To run this example:
 - Create two projects with the same code
 - One MIDlet should listen for messages (server) on port 1234 and another on port 1235 – different ports
 - Run them on two different (emulated) phones.

Exchanging Messages



SMSMidlet

```
private void startReceive() {
    if (mReceiver != null)
        return;
    // start receive thread
    mReceiver = new Thread(this);
    mReceiver.start();
}

private boolean mEndNow = false;
private MessageConnection conn = null;

public void run() {
    Message msg = null;
    String msgReceived = null;
    conn = null;
    mEndNow = false;
    /** Check for sms connection. */
    try {
        conn = (MessageConnection) Connector.open("sms://:" + mPort);
        msg = conn.receive();
        while ((msg != null) && (!mEndNow)) {

            if (msg instanceof TextMessage) {

                msgReceived = ((TextMessage)msg).getPayloadText();
                if (msgReceived.equals("red")) {
                    Display.getDisplay(this).callSerially(new SetRed());
                } else if (msgReceived.equals("blue")) {
                    Display.getDisplay(this).callSerially(new SetBlue());
                }
            }

            msg = conn.receive();
        }
    } catch (IOException e) {
        // normal exit when connection is closed
    }
}
```

SMSMIDlet uses a separate receive thread. The `startReceive()` method contains the code that starts the receive thread.

The `run()` method contains the logic of the receive thread. The blocking `receive()` call is used in this case.

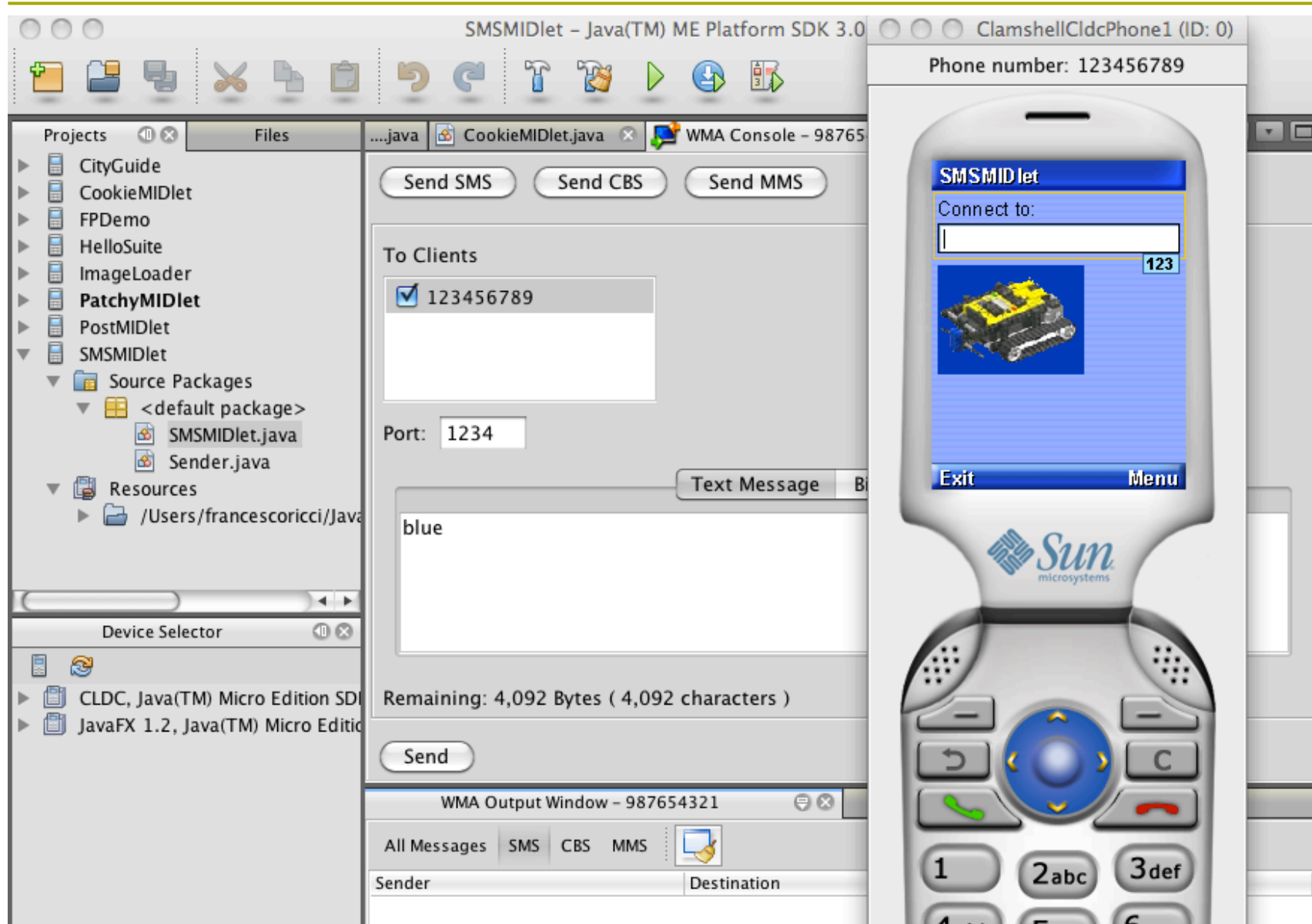
[SMSMIDlet.java](#)

[Sender.java](#)

Testing SMSMIDlet with WTK 2.x WMA Console

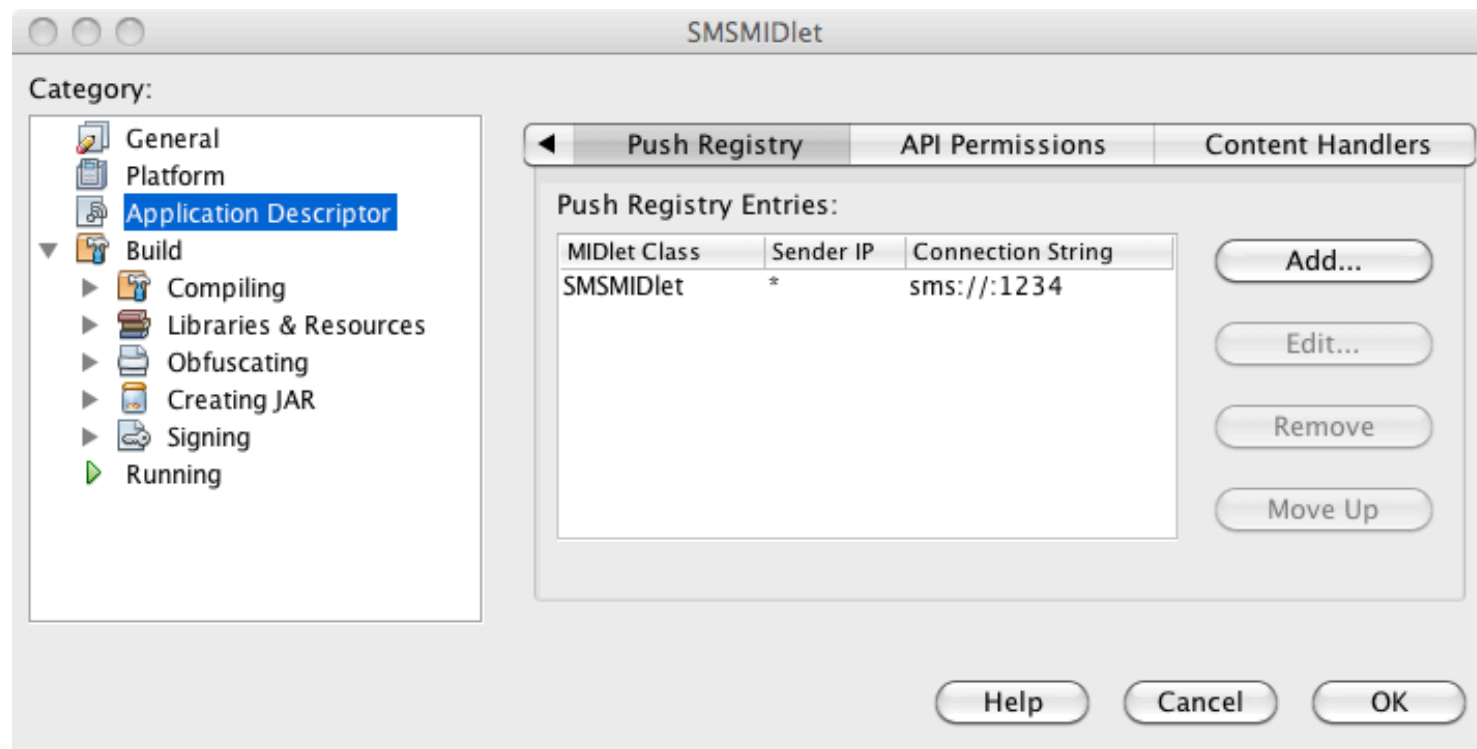
- ❑ You can test `SMSMIDlet` using a utility called the WMA console. To access the WMA console, select Tools → WMA Console.
- ❑ To test Start `SMSMIDlet`, first start an instance of the emulator. The caption contains the phone number assigned to the emulator (123456789 by default)
- ❑ Start the WMA console, select the address of the emulator 123456789, enter **1234** into the Port box, and then enter **red** for the message and click the Send button. Note that the `SMSMIDlet` instance now displays an image with a red background
- ❑ Instead to start the WMA console you can also start two emulator instances of `SMSMIDlet` and send SMS messages between them (*but they should listen on different ports*).

WMA Console



SMS for launching applications

- ❑ As we shown for the socket connection
- ❑ A midlet can be started when an SMS is received
- ❑ It must be deployed via OTA



Receiving CBS Messages

- ❑ **Cell Broadcast Service**, or CBS is a carrier version of SMS - it enables a cell phone operator to broadcast messages to a group of cell phone users
- ❑ A WMA application can only receive CBS messages, so the `MessageConnection` for CBS can only be opened in the server (receiver) mode
 - `conn = (MessageConnection) Connector.open("cbs://:12345");`
 - `msg = conn.receive(); // Blocking for message`
- ❑ The digits `12345` serves as a message identifier, they may represent different information channels that you can subscribe to
- ❑ **In Italy only the channel "50" is available.** On this channel you should receive information about cell area every 10 seconds.

See also...

- HTTP MIDlet examples

<http://www.java2s.com/Code/Java/J2ME/HttpMIDlet.htm>

- MIDP Network Programming using HTTP and the Connection Framework

<http://developers.sun.com/techttopics/mobility/midp/articles/network/index.html>

- The Wireless Messaging API 2.0

<http://developers.sun.com/techttopics/mobility/midp/articles/wma2/>

- WMA 2.0 Specification

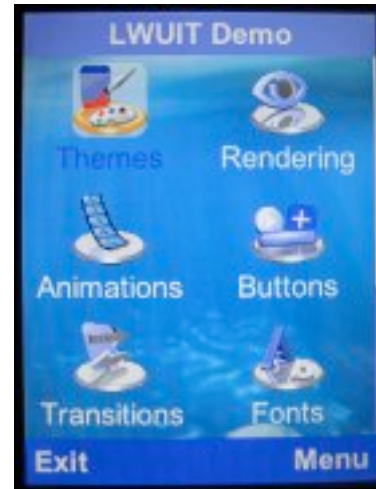
<http://jcp.org/aboutJava/communityprocess/final/jsr120/index2.html>

Lightweight UI Toolkit (LWUIT)

- ❑ Lightweight library bundled with the application
- ❑ Compelling UI, consistent across platforms (CLDC, CDC, Java SE)
- ❑ Minimal requirements
 - CLDC 1.1 + MIDP 2.0 or
 - CDC + PBP
- ❑ Highly portable and open source
- ❑ Inspired by Swing, optimized for mobile and embedded
- ❑ Tools support: Resource editor, NetBeans



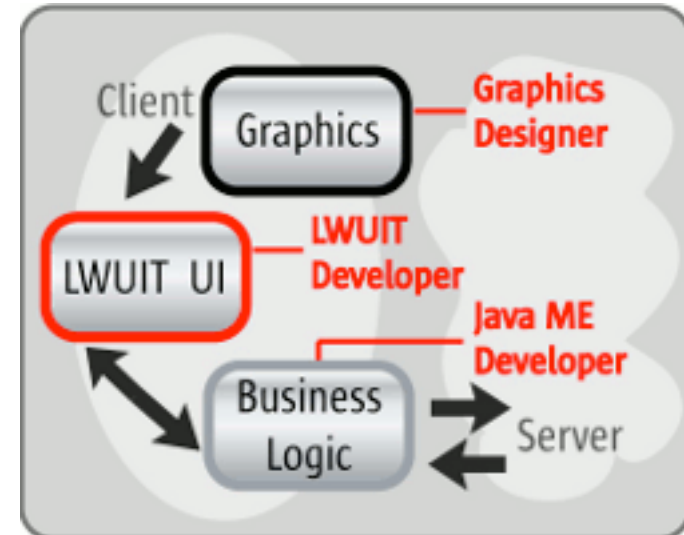
Better adaptation to screen size



- ❑ LWUIT Demo application
- ❑ Identical application binary file (with no built-in device-specific knowledge) running on three entirely different Java ME platforms:
 - Java ME SDK 3.0 Mobile Emulator
 - Mid-range Sony Ericsson G705
 - HTC Diamond touch screen device

LWUIT features

- ❑ LWUIT provides a clean separation between the UI development, the graphics design, and the business logic
- ❑ LWUIT is based on the **MVC** (model-view-controller) paradigm
- ❑ LWUIT deployment: bundle the LWUIT library and resources with the application - the LWUIT components become part of the application deployment unit and are downloaded and deployed transparently when the user installs the application on their device.



Hello Word

```
import javax.microedition.midlet.*;
import com.sun.lwuit.*;
import com.sun.lwuit.events.*;

public class HelloLWUITMidlet extends MIDlet implements ActionListener {
    public void startApp() {
        Display.init(this);
        Form f = new Form("Hello, LWUIT!");
        f.show();

        Command exitCommand = new Command("Exit");
        f.addCommand(exitCommand);
        f.addCommandListener(this);
        //f.setCommandListen(this);
    }

    public void pauseApp() {}
    public void destroyApp(boolean unconditional) {}
    public void actionPerformed(ActionEvent ae) {
        notifyDestroyed();
    }
}
```

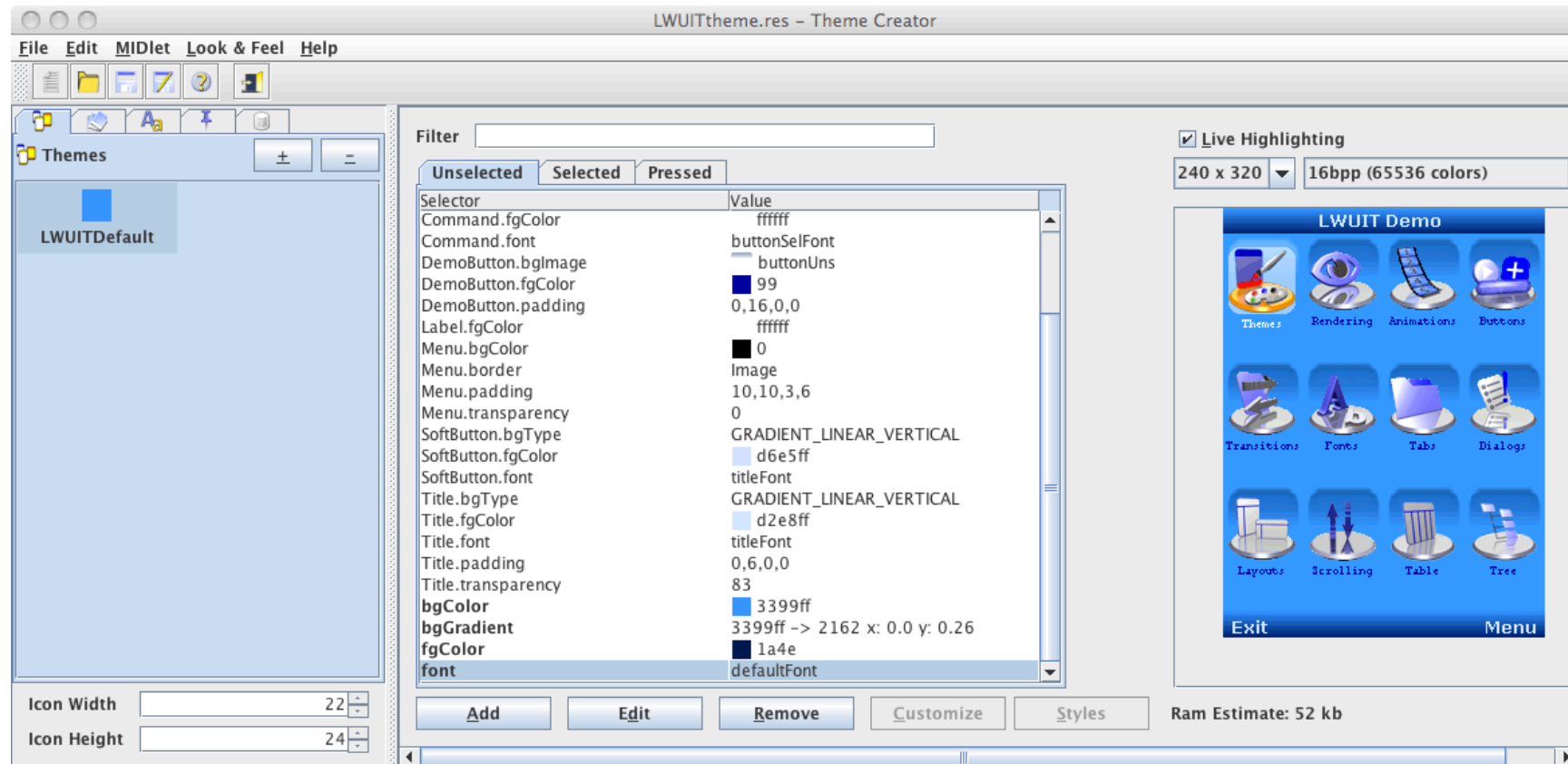


[code](#)

Key Features

- ❑ **Rich Widgets:** label, button, text box, input field, list, table, calendar, tree, spinner, virtual keyboard, HTML renderer, and more
- ❑ **Layouts Manager:** flexible and powerful if the application runs on different screen sizes
- ❑ **Pluggable Look and Feel & Themes:** the Theme Creator toolkit lets you create a CSS-like file that can be loaded or changed at runtime, controlling the look and feel of the application
- ❑ **Fonts:** The library features bitmap fonts and a tool that lets you create fonts on your desktop
- ❑ **Touch Screen:** All LWUIT components support touch events - no special coding is needed
- ❑ **Animations & Transitions:** Various visual effects that are available out-of-the-box
- ❑ **3D and SVG Graphics** integration
- ❑ **Lightweight:** low memory footprint and processing requirements, adapts to platform.

Theme Creator



- ❑ In the LWUIT distribution is called ResourceEditor.jar

Demos

- In the distribution zip file (<http://www.oracle.com/technetwork/java/javame/tech/lwuit-141954.html>) there are 3 demos and a user guide



- LWUIT-Makeover, LWUITBrowser, LWUITDemo

Resources for LWUIT

- Project home page: <https://lwuit.dev.java.net/>
- Introductions and tutorial:
 - http://java.sun.com/developer/technicalArticles/javame/lwuit_v1_3/
 - <https://lwuit.dev.java.net/nonav/tutorial/>
- A more complex sample application and tutorial
 - <https://lwuit.dev.java.net/servlets/ProjectProcess?tab=3>

Captures Traffic Data Using GPS-Enabled Cell Phones

- ❑ **Nokia** and University of California, **Berkeley** researchers are developing technology: **using GPS-enabled mobile phones to monitor real-time traffic flow**
- ❑ One hundred vehicles were deployed on a 10-mile stretch of highway
- ❑ Each car was equipped with a GPS-enabled mobile phone that **periodically sent anonymous speed readings** to servers that computed traffic conditions
- ❑ Traffic information was **displayed on the Internet**, allowing viewers to see traffic in real time
- ❑ GPS-based systems are capable of **pinpointing a car's location within a few meters** and calculating traveling speed within 3 miles per hour
- ❑ The researchers say that using GPS-equipped cell phones to monitor traffic could help provide information on everything from multiple side-street routes in urban areas to hazardous driving conditions or accidents on rural roads.

http://www.berkeley.edu/news/media/releases/2008/02/08_gps.shtml