Utilizando Bluetooth GPS diretamente em Java ME

Este artigo mostra como utilizar um GPS externo através de Bluetooth, com Java ME.

Um dos acessórios mais vendidos para telefones móveis e PDAs são os módulos Bluetooth GPS (BT-GPS). Esses dispositivos recebem informações de posicionamento do sistema de satélites GPS e permitem verificar sua posição atual com margem de cerca de 5m de precisão. É possível acessar esses dados via Bluetooth a partir de telefones móveis, e nestes, via APIs Java.

JSR 179 - Location API

Em dispositivos equipados com a implementação da JSR 179 - Location API, não há necessidade de uso da API Bluetooth (JSR 82), pois a Location API se encarrega de todos os passos necessários à obtenção da posição, tais como:

- Busca do dispositivo GPS Bluetooth na área local
- Pareamento (pairing) com o dispositivo
- Obtenção de informação de dados de posicionamento
- Parsing das strings NMEA utilizadas na comunicação dos dados de posicionamento, retornando objetos de alto nível em Java.

Para usar GPS através da JSR 179, baixe o documento MIDP: Location API Developer's Guide v2.0, que também contém código-fonte e uma aplicação pronta para testar.

Clique [1] aqui para fazer uma busca de todos os aparelhos que suportam a JSR 179 - Location API 1.0.

JSR 82 - Bluetooth API

Caso seu dispositivo não implemente a JSR 179, você poderá utilizar um módulo Bluetooth GPS e ler as informações de posicionamento manualmente. Para tanto, é preciso implementar as seguintes tarefas:

1. Pesquisar o dispositivo Bluetooth GPS
2. Conectar ao dispositivo GPS
3. Ler e interpretar as sentenças NMEA que contém informações de posicionamento

Para o primeiro passo necessitamos buscar por um dispositivo Bluetooth que implemente o serviço RFCOMM. Uma classe que realiza todo este trabalho, "BTManager.class", está incluída no código-fonte completo deste exemplo.

Se você necessita de mais informações sobre busca de dispositivos, veja Como encontrar dispositivos e serviços Bluetooth.

Para o propósito de ler dados do BT-GPS vamos criar uma classe chamada GpsBt. A primeira coisa a se fazer é criar uma variável para armazenar a URL do seu dispositivo.

```
// dispositivo bluetooth corrente
public String btUrl = "";
public String btName = "";

public void setDevice(String btUrl, String btName) {
  this.btUrl = btUrl;
  this.btName = btName;
}
```

Agora precisamos nos conectar ao dispositivo e começar a ler os dados:

```
  public void start() {
  if (isActive) {
    stop();
  }
  connect();
  if (isConnected) {
    isActive = true;
    Thread t = new Thread(this);
```


(C) Copyright Nokia 2014. All rights reserved.

```
      t.start();
    }
  }

  public void connect() {
    if (btUrl == null || (btUrl.trim().compareTo("") == 0)) {
      isConnected = false;
      return;
    }
    try {
      conn = (StreamConnection) Connector.open(btUrl, Connector.READ_WRITE);
      in = new DataInputStream(conn.openInputStream());
      isConnected = true;
      mode = 0;
    } catch (IOException e) {
      close();
    }
  }

  public void run() {
    isActive = true;
    while (isActive) {
        // checa se a conexão ainda está aberta
        if (!isConnected && isActive) {
          // conecta ao dispositivo GPS
          connect();
        } else {
          // lê strings NMEA
          readNMEASentences();
        }
    }
    close();
    isActive = false;
  }
```

Como você pode ter notado, estou implementando o loop de leitura usando uma Thread. A razão para isto é que o dispositivo GPS está sempre mandando dados, então você precisa lê-los continuamente para impedir que o buffer da conexão seja maior que a sua capacidade. Os dados enviados pelo dispositivo GPS são sentenças NMEA, que fornecem várias informações sobre o status da posição atual. O que estamos procurando é pela sentença GPGGA, que nos dá informações essenciais sobre localização de nosso aparelho.

```
  public void readNMEASentences() {
    try {
      if (!isConnected) {
        return;
      }
      // checa os caracteres disponíveis
      int size = in.available();
      if (size <= 0) {
        return;
      }
      // lê dados
      for (int j = 0; j < size; j++) {
        int i = in.read();
        if (i != -1) {
          char l = (char) i;
```


(C) Copyright Nokia 2014. All rights reserved.

```
          switch (mode) {
          case (STATE_SEARCH_SENTENCE_BEGIN): {
            // procura pelo início da sentença
            if (l == '$') {
              // início da sentença encontrada
              mode = 1;
              sb.setLength(0);
            }
          }
            break;
          case (STATE_READ_DATA_TYPE): {
            // checa qual tipo de sentença temos
            sb.append(l);
            if (sb.length() == 6) {
              if (sb.toString().startsWith("GPGGA")) {
                mode = STATE_READ_SENTENCE;
                sb.setLength(0);
              } else {
                mode = STATE_SEARCH_SENTENCE_BEGIN;
                sb.setLength(0);
              }
            }
          }
            break;
          case (STATE_READ_SENTENCE): {
            // lê os dados da sentença
            sb.append(l);
            if ((l == 13) || (l == 10) || (l == '$')) {
              mode = STATE_SEARCH_SENTENCE_BEGIN;
              currentInfo = new String(sb.toString());
            }
          }
            break;
          }

        } else {
          close();
        }
      }

    } catch (Exception e) {
      close();
    }
  }
```

Após termos obtido a sentença correta precisamos apenas interpretar a informação e mostrá-la ao usuário:

```
  public Location getLocation() {
     Location location = new Location();
     if (isConnected && isActive && currentInfo != null) {
         location.parseGPGGA(currentInfo);
     }
     return location;
   }
```


(C) Copyright Nokia 2014. All rights reserved.

Para nos ajudar a buscar a informação, utilizamos uma classe, chamada Location, para interpretar a sentença GGA e encapsular os dados de posicionamento. Também estamos utilizando a classe StringTokenizer para ler todos os tokens presentes na sentença.

```
public class Location {

  // Elementos NMEA GPGGA
  String utc;
  String latitude;
  String northHemi;
  String longitude;
  String eastHemi;
  String altitude;
  int quality;
  int nSat;
  String horDilution;
  String altitudeUnit;
  String geoidalHeight;
  String geoidalHeightUnit;
  String diffCorrection;
  String diffStationId;

  /**
   * Método para quebrar strings NMEA e retorna a localização. Para mais informações,
observe esta página
   * http://www.gpsinformation.org/dale/nmea.htm#GGA
   *
   * @param value -
   *          string que representa uma string NMEA GGA
   */
  public void parseGPGGA(String value) {
    // Classe utilitária para quebrar strings
    StringTokenizer tok = new StringTokenizer(value, ",");

    utc = tok.nextToken();
    latitude = tok.nextToken();
    northHemi = tok.nextToken();
    longitude = tok.nextToken();
    eastHemi = tok.nextToken();
    quality = Integer.parseInt(tok.nextToken());
    nSat = Integer.parseInt(tok.nextToken());
    horDilution = tok.nextToken();
    altitude = tok.nextToken();
    altitudeUnit = tok.nextToken();
    geoidalHeight = tok.nextToken();
    geoidalHeightUnit = tok.nextToken();
    diffCorrection = tok.nextToken();
    diffStationId = tok.nextToken();
  }
}
```

Em suma, agora temos uma classe capaz de ler sentenças NMEA de um BT-GPS e obter dados de posicionamento. Veja o código-fonte para uma aplicação completa de posicionamento.

Downloads

(C) Copyright Nokia 2014. All rights reserved.

- Código-fonte
- Arquivo JAD
- Arquivo JAR

Referências

- Vietnamese GPS - Intelligent on the go
- NMEA Reference

(C) Copyright Nokia 2014. All rights reserved.