

**GUJ**

Notícias, tutoriais, e o maior fórum brasileiro sobre Java.

Artigo – GUJ.com.br

Diamond Powder - um framework Java ME para coleta de dados

Autor

Renato Bellia: é formado em Engenharia de Computadores, trabalha com Java EE a mais de 08 anos, e atualmente está utilizando Java ME em seus projetos.

Gravata

[Este artigo explora alguns recursos do Diamond Powder, um framework Open Source para Java ME, que facilita a vida do desenvolvedor de MIDlets. Desenvolva com rapidez formulários para coleta de dados em seus MIDlets, e ganhe em flexibilidade e em gerenciamento automático de persistência RMS.](#)

Introdução

O desenvolvedor de MIDlets tem que saber lidar com LCDUI e RMS - duas APIs fundamentais do profile MIDP. Enquanto LCDUI permite a construção de interfaces simples, RMS permite o armazenamento de dados persistentes no dispositivo. Muitas vezes uma aplicação MIDlet, dentre outras responsabilidades, tem que implementar funcionalidades de coleta de dados. Por coletor de dados vamos entender um componente de software que, ajuda o usuário a tomar notas rápidas de informações, além de gerenciar o armazenamento e recuperação posterior de tais informações. É possível desenvolver coletores de dados com LCDUI e RMS. Mas isso é um trabalho repetitivo – de um projeto com coletor de dados a outro mudam os nomes dos campos da interface com o usuário, mas a lógica de programação envolvida será praticamente a mesma.

Diamond Powder é um framework open source para Java ME que acelera a criação de coletores de dados em aplicações baseadas em MIDlets. Ele nasceu durante a implementação um projeto que oferece uma interface móvel para os usuários, via MIDlet. Tal MIDlet demandava não um, mas três coletores de dados. Um princípio que eu procuro seguir sempre é o “DRY”, e significa “Don’t Repeat Yourself”, ou “não se repita”, em português. Eu percebi que teria muito trabalho repetitivo para concluir este MIDlet, e a idéia do Diamond Powder começou a se formar. Na prática um coletor de dados gerencia uma seqüência de formulários, com campos de texto, calendários, listas de opções e telas de help. Para descrever um coletor de dados o Diamond Powder utiliza um formato de configuração bastante simples e direto – um programador que conhece LCDUI vai se sentir em casa, pois a configuração remete a vários elementos desta API. Vamos verificar como utilizar esse formato de configuração, batizado de “schema”, e como incorporar os coletores de dados em seus MIDlets.

Um cenário simples

Imagine o desenvolvimento de desenvolve um MIDlet que ajude motoristas de carro a manter registros sobre o consumo de combustível de seu carro: O motorista abastece no posto de gasolina e saca seu dispositivo móvel para tomar a nota sobre o valor do odômetro, a quantidade de combustível, o preço do combustível, o nome do posto de gasolina, e a data atual. Mais tarde o motorista precisa recuperar tais dados. Este é um coletor de dados, e o Diamond Powder pode ajudar neste desenvolvimento.

Sem dúvida, o MIDlet poderia ir além, fazendo alguma matemática com tais dados, desenhando gráficos, enviando os registros pela internet, etc, mas estas funcionalidades não competem a um coletor de dados.

Definindo um Schema

Um “schema” do Diamond Powder descreve um coletor de dados. Veja alguns elementos que podem ser declarados em um schema:

- schema: é o elemento “raiz”, e contém um nome e um número de versão;
- flow: ou fluxo, é uma seqüência de páginas (page) por onde o usuário pode navegar, durante uma coleta de dados – todo schema deve conter ao menos um flow;
- page: é uma etapa de um fluxo de coleta de dados, é um agrupador de campos de entrada de dados;
- help: é um texto de ajuda associado a uma página;
- field: é um campo de entrada de dados, semelhante aos itens LCDUI: textfield, datefield, choicegroup, stringitem, e um campo especial chamado ‘filter’;
- listmodel: é uma lista de elementos para os choicegroups, onde cada elemento está associado a um número identificador.

No código fonte de seu MIDlet o schema se concretiza com um Hashtable – veja a listagem 1.

Construindo um Coletor de Dados

A classe Collector da API do Diamond Powder é uma subclasse de Form, e deve ser instanciada recebendo o objeto de Display do MIDlet, o schema e o nome do flow que será seguido. Além disso:

- O coletor deve receber via addCommand pelo menos um comando OK e um comando BACK para se integrar com os demais dos formulários do MIDlet.
- O coletor deve receber via setCommandListener o tratador de eventos do MIDlet (o próprio MIDlet neste exemplo)
- Finalmente o método showCurrentPage deve ser invocado para a montagem interna das páginas do coletor (ainda não será exibido no display).

Veja a listagem 2.

Os comandos OkCommand1 e BackCommand utilizados neste exemplo devem possuir tratamento de eventos para navegar para outros formulários do seu MIDlet que convivem com o coletor. Veja um exemplo na figura 1.

Exibindo o Coletor de Dados

Para exibir o coletor deve fazê-lo como se estivesse exibindo um formulário comum. Veja a listagem 3.

As figuras 2 e 3 mostram o coletor em funcionamento.

Persistência Automática

Para armazenar e restaurar todos os dados capturados por um coletor utilizamos a classe StorageManager. Acompanhe a listagem 4.

Obtendo Dados de um Coletor

A classe collector oferece o método getFields(), que retorna uma Hashtable contendo os objetos LCDUI interpretados a partir do schema.

A listagem 5 mostra como obter do coletor o preço do combustível.

Poupando Trabalho Repetitivo

Se este coletor de dados fosse construído diretamente com LCDUI e RMS, teríamos algo em torno de umas 100 linhas de código. Fica por conta do leitor a experiência: alterar o schema (listagem 1) acrescentando novos campos e páginas, e calcular o tempo e esforço poupados.

Coletores são Dinâmicos

Um ponto central do Diamond Powder é que um coletor analisa um schema e monta os campos para entrada de dados em tempo de execução. Neste exemplo o schema está "hard-coded", mas nada impede a construção dinâmica do Hashtable de schema.

Se uma alteração no coletor de dados for necessária (novos campos, por exemplo), é perfeitamente possível entregar uma nova versão de um schema em tempo de execução (via SMS, HTTPConnection, ...) para o MIDlet, dispensando o redeployment deste último. Tal característica confere um novo grau de liberdade ao desenvolvedor de MIDlets.

Conclusões

O Diamond Powder nasceu da necessidade de simplicidade e agilidade na construção de coletores de dados em aplicações Java ME. Verificamos através dos exemplos como construir rapidamente um schema de coleta de dados, e como integrar um coletor em sua aplicação MIDlet. Existem outros recursos disponíveis e novidades em implementação que podem ser acompanhados através do blog do projeto.

Para Saber Mais
http://diamond-powder.blogspot.com - o blog é em inglês, mas aceitamos comentários em português.
https://diamond-powder.dev.java.net - site do projeto, com bibliotecas e exemplos para download.

Listagem 1. Definição de um schema de coleta de dados

```
public Hashtable getSchema() {
    Hashtable schema = new Hashtable();
    //schema declaration: name;version
    schema.put("schema", "fuelControl;1");

    //flow declaration: page1;page2;...
    schema.put("flow.basicRecord", "numbers;extra");

    //page declaration: title;field1;field2;...
    schema.put("page.numbers", "The Numbers;odometer;fuelAmount;fuelPrice");
    schema.put("page.extra", "Gas Station;gasStationName;gasStationBrand");

    //help for page: help text
    schema.put("help.extra", "Enter the gas station name and brand");
    schema.put("help.numbers",
        "Enter the odometer mark, the supplied fuel amount and the fuel price");

    //text field declaration: field type;label;size;mask
    schema.put("field.odometer", "textfield;odometer;6;numeric");
    schema.put("field.fuelAmount", "textfield;fuel amount;5;decimal");
    schema.put("field.fuelPrice", "textfield;fuel price;5;decimal");
    schema.put("field.gasStationName", "textfield;gas station;40;initial_caps_word");

    //choice gorup declaration: field type;label;list model;mode
    schema.put("field.gasStationBrand", "choicegroup;brand;allBrands;exclusive");
    //list model declaration: value1;label1;value2;label2;...
    schema.put("listmodel.allBrands", "999;undefined;1;Atlantic;2;Chevron;3;Esso;4;Texaco");
    return schema;
}
```

Listagem 2. Construindo um coletor de dados

```
private Collector collector;
public Collector getCollector() {
    if (collector == null) {
        collector = new Collector(getDisplay(), getSchema(), "basicRecord");
        collector.setTitle("collector");
        collector.addCommand(getOkCommand1());
        collector.addCommand(getBackCommand());
        collector.setCommandListener(this);
        collector.showCurrentPage();
    }
    return collector;
}
```

Listagem 3. Exibindo um coletor de dados

```
getDisplay().setCurrent(getCollector());
```

Listagem 4. Armazenamento e recuperação

```
// um StorageManager deve referenciar um coletor
StorageManager sm = new StorageManager(getCollector());

// armazena dados em registro RMS, e retorna o id do registro
int id = sm.store();

// para limpar o collector
getCollector().reset();

// recupera os dados armazenados em RMS, com o id especificado
sm.load(id);

// altera um registro RMS com os dados disponíveis no coletor
sm.update(id);

// remove um registro, dados o schema e o id
StorageManager.delete("fuelControl", id);
```

Listagem 5. Obtendo campos do coletor

```
Object field = getCollector().getFields().get("field.fuelPrice");
String strPreco = ((TextField) field).getString();
```

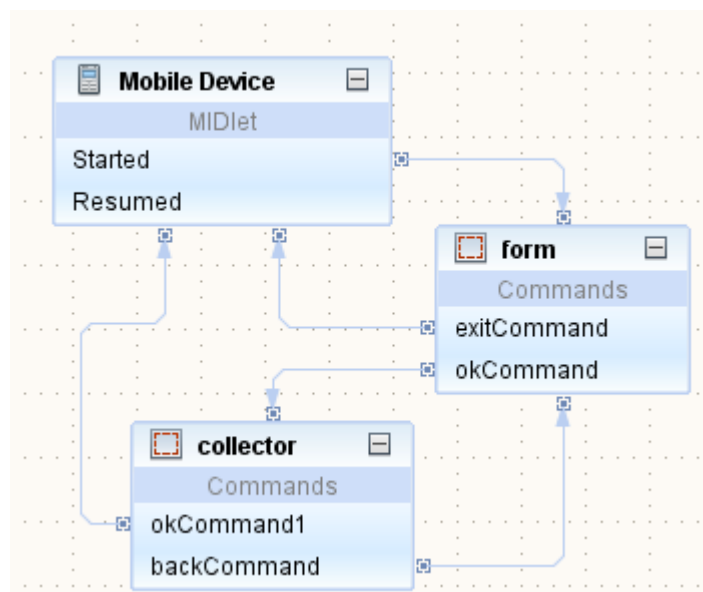


Figura 1. O coletor e os demais formulários de um MIDlet.

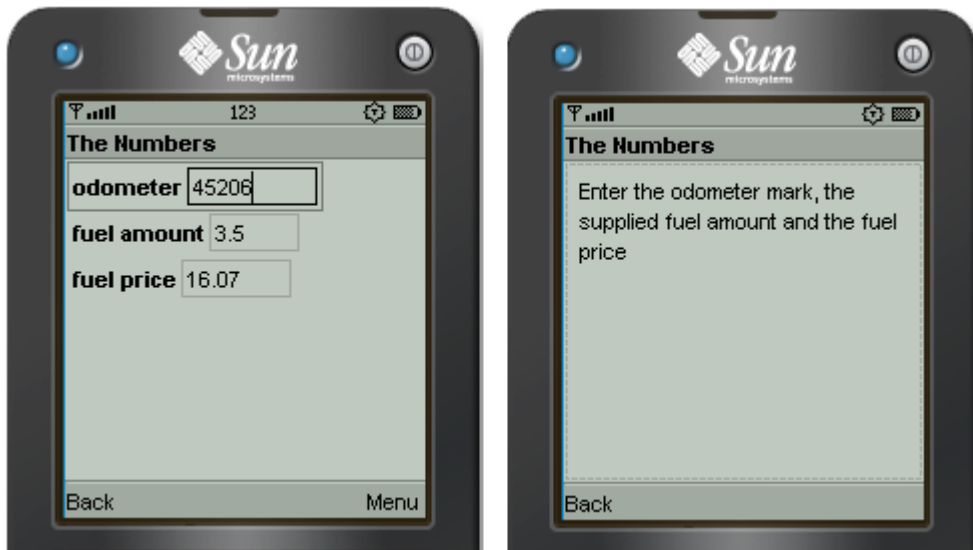


Figura 2: página 'numbers' do fluxo 'basicRecord', e o respectivo help.

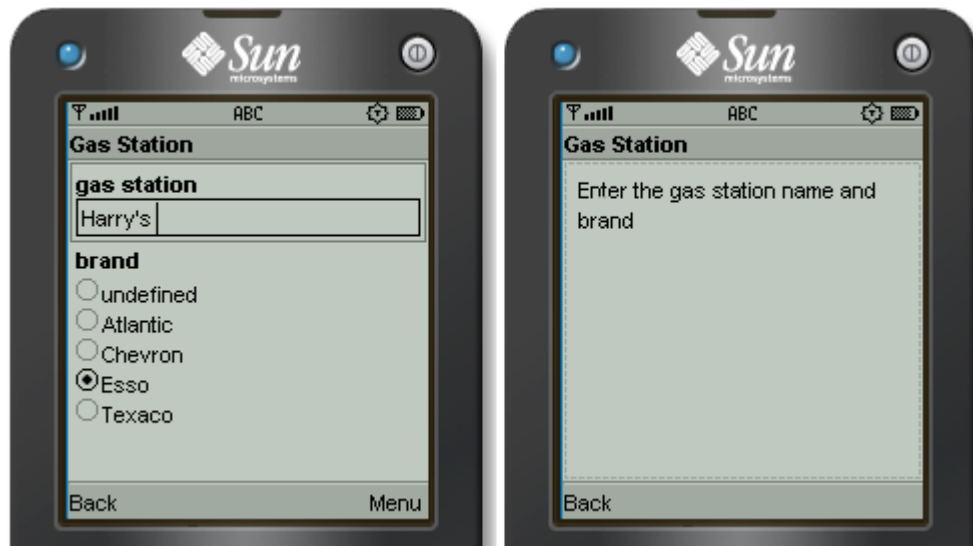


Figura 3: página 'extra' do fluxo 'basicRecord', e o respectivo help.