High Level UI using List

Version 0.2, Draft

INFORMATION GUIDE

COPYRIGHT

Trademarks and Service Marks

About This Document

This document gives an overview of High Level class *List* and provides a sample code snippet explaining the implementation of class *List*.

Scope

This document is intended for MIDP developers wishing to develop mobile Java ME applications. It assumes good knowledge of java programming language.

Document History:

Date	Version	Comment
04/02/09	0.2	Draft

Reference:

1. MIDP 2.0 Specification:

http://jcp.org/en/jsr/detail?id=118

Abbreviations:

MIDP	Mobile Information Device Profile
UI	User Interface

Table of Contents

Introduction 5
Overview 5
EXCLUSIVE 5
MULTIPLE 6
IMPLICIT 6
Creating List 7
List Operations 7
Sample code snippet for the List 8

Table of Figures

Figure 1: Exclusive List 5
Figure 2: Multiple List 6
Figure 3: Implicit List 7

Introduction

User can interact with the device through Screen. Screen combines and organizes graphics objects and manages user inputs through the device. Screen is represented by the *javax.microedition.lcdui.Screen* object. One of the Screen types is List.

It has a common behavior like *javax.microedition.lcdui.ChoiceGroup*; scrolling and traversing but it also contains additional API. This document focuses on *List* and its importance in developing a MIDlet application.

Overview

javax.microedition.lcdui.List is an independent *Displayable* object that allows user to choose between different elements. These elements consist of simple string, user can also include image per element as well. *List* has a behavior common to ChoiceGroup; scrolling, navigating, selecting and deselecting item appropriately.

List can be created in two different modes:

Single Choice: In this mode, only one element can be selected at a time from the List.

Multiple Choice: In this mode, one or more than one element can be selected at a time from the List.

List implements *javax.microedition.lcdui.Choice* Interface to differentiate between these two modes. This is done by providing the following Choice types:

EXCLUSIVE

The EXCLUSIVE choice type is used to select only one element at a time. By default, one element is selected at any given time. *List* implementation uses Radio Buttons for visual representation to the user. Figure 1 shows the EXCLUSIVE type ChoiceGroup.

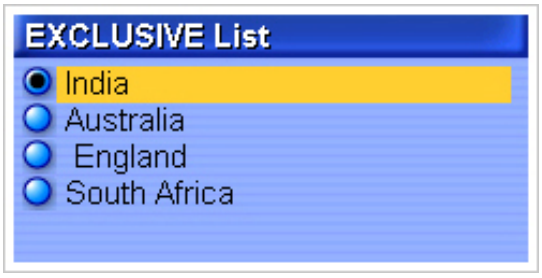

Figure 1: Exclusive List

Applications for which the selected element is significant should set the selection explicitly. There is no way for the user to deselect an element within an EXCLUSIVE Choice. Selection in this choice type is done by selecting the target element and deselecting the previous selected element. EXCLUSIVE type List must register at least one Command object, otherwise no events will be sent to the command listener. This is because the EXCLUSIVE type List does not trigger any events as the user interacts with the list.

MULTIPLE

The MULTIPLE choice type is used to select one or more than one element in any combination. List implementation uses Checkbox for visual representation to the user. Figure 2 shows MULTIPLE type List. The select operation toggles the selected state of an element, leaving the selected state of other elements unchanged. MULTIPLE type List must register at least one Command object, otherwise no events will be sent to the command listener. This is because the MULTIPLE type List does not trigger any events as the user interacts with the list.

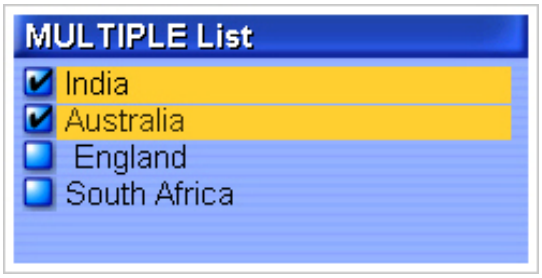

Figure 2: Multiple List

IMPLICIT

IMPLICIT type List is similar to EXCLUSIVE; the only difference is the selection of an element. When the user selects an item from an IMPLICIT mode list, the list notifies its command listener using the special command object defined as *List.SELECT_COMMAND*. This object gets passed to the listener's *commandAction(Command c, Displayable d)* method as the first argument whenever a new item is selected. In other words, you check for implicit selection using code like this:

```
public void commandAction( Command c, Displayable d ){
    if( c == List.SELECT_COMMAND ){
        /* implicit selection...*/
    }
}
```

Figure 3 shows the IMPLICIT type List.

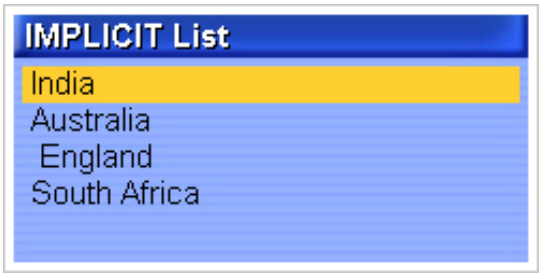

Figure 3: Implicit List

Creating List

List can be created using one of the two constructors.

```
List (String label, int listType)
```

The *List* constructor requires at least a label and a type value. Additionally, a String array and an Image array containing the elements can be passed to the constructor.

```
List(String label, int list Type,String[] stringElements, Image[] imageElements)
```

where

label -- indicates the List name.

choiceType – this variable specifies the Choice for List; EXCLUSIVE, MULTIPLE or IMPLICIT

stringElements --set of strings specifying the string parts of the List elements.

imageElements--set of images specifying the image parts of the List elements.

List Operations

Some of the important List operations are as follows:

- Element can be added to the List using *append(String stringpart, Image imagepart)* where stringpart is compulsory and imagepart can be null if no image is required to be added to the element.

- Element can be inserted using *insert(int index, String stringpart, Image imagepart)* where stringpart and imagepart are same as adding an element. Index signifies the index of the element where insertion should occur.
- Element can be set using *set(int index, String stringpart, Image imagepart)* where stringpart and imagepart are same as inserting an element.

- Element can de deleted using *delete(int elemNum)* where elemNum signifies the index of the element to be deleted. All elements from ChoiceGroup can be deleted using *deleteAll()* method.
- Element can be retrieved using *getString(int elemNum) , getImage(int elemNum)* to get String and Image respectively by passing the elemNum. Similarly index of an element can be retrieved using *getSelectedIndex()*.
- Element can be selected using *setSelectedIndex(int elemNum, boolean selected)* where elemNum refers to the index. The selected parameter must be true or false to select or deselect an element.
- To check whether the element is selected *isSelected(int elemNum)* method can be used.
- Total elements present in a ChoiceGroup can be determined using *size()* method.

Sample code snippet for the List

The sample code given below shows how to use *List* class.

Class: ListMidlet.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Choice;

public class ListMidlet extends MIDlet implements CommandListener {

   private static final String EXIT = "Exit";
   private static final String SHOW = "Show";
   private static final String BACK = "Back";
   private static final int CHOICE_EXCLUSIVE = 0;
   private static final int CHOICE_MULTIPLE = 1;
   private static final int CHOICE_IMPLICIT = 2;
   private static final String LIST_TITLE = "Selection List";
   private final Command CMD_EXIT = new Command(EXIT, Command.EXIT, 1);
   private final Command CMD_SHOW = new Command(SHOW, Command.SCREEN, 2);
   private final Command CMD_BACK = new Command(BACK, Command.SCREEN, 3);
   private Display display = null;
```

```
private List selectionList = null;
private List selectedList = null;
private static final String cricketteam[] = {"India", "Australia", " England", "South Africa"};
private static final String choiceType[] = {"EXCLUSIVE List", "MULTIPLE List",
  "IMPLICIT List"};
private boolean isDone;

public ListMidlet() {
    if (!isDone) {
        init();
        isDone = true;
    }
}

protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
}

protected void pauseApp() {
}

protected void startApp() throws MIDletStateChangeException {
    selectionList.setCommandListener(this);
    display.setCurrent(selectionList);
}

public void commandAction(Command cmd, Displayable dis) {
    if (cmd == CMD_EXIT) {
        try {
            destroyApp(true);
            notifyDestroyed();
        } catch (MIDletStateChangeException e) {
            e.printStackTrace();
        }
    } else if ((cmd == CMD_SHOW) && (dis == selectionList)) {
        switch (selectionList.getSelectedIndex()) {
            case CHOICE_EXCLUSIVE:
                selectedList = new List(selectionList.getString(selectionList.getSelectedIndex()),
                        Choice.EXCLUSIVE, cricketteam, null);
                break;

            case CHOICE_IMPLICIT:
                selectedList = new List(selectionList.getString(selectionList.getSelectedIndex()),
                      Choice.IMPLICIT, cricketteam, null);
                break;

            case CHOICE_MULTIPLE:
                selectedList = new List(selectionList.getString(selectionList.getSelectedIndex()),
```

```
                    Choice.MULTIPLE, cricketteam, null);
                break;
            }
            selectedList.addCommand(CMD_EXIT);
            selectedList.addCommand(CMD_BACK);
            selectedList.setCommandListener(this);
            display.setCurrent(selectedList);
        } else if ((cmd == CMD_BACK) && (dis == selectedList)) {
            selectedList = null;
            display.setCurrent(selectionList);
        }
    }

    public void init() {
        display = Display.getDisplay(this);
        selectionList = new List(LIST_TITLE, Choice.EXCLUSIVE, choiceType, null);
        selectionList.addCommand(CMD_EXIT);
        selectionList.addCommand(CMD_SHOW);
    }
}
```