Showing calendar events in Java ME

Overview

This code snippet demonstrates how to get a list of calendar events using the PIM API, show it to the user, and show the values of all fields of the selected event.

Source file: ShowingCalendarEvent.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Alert;

import javax.microedition.pim.PIM;
import javax.microedition.pim.PIMItem;
import javax.microedition.pim.EventList;
import javax.microedition.pim.Event;
import javax.microedition.pim.PIMException;

import java.util.Enumeration;
import java.util.Vector;
import java.util.Date;


public class ShowingCalendarEvents extends MIDlet implements CommandListener {

    private Display display;

    // List where events will be placed.
    private List eventListCtrl;
    // Form where details of chosen event will be placed.
    private Form eventForm;

    // Command for showing details of chosen event. Placed in eventListCtrl.
    private Command cmdShowDetails;
    // Command for returning back from detailsForm to eventListCtrl.
    private Command cmdBack;
    // Command for exiting from application.
    private Command cmdExit;

    // Contains events retrieved from PIM events array.
    private Vector eventsArray;

    /**
     * Constructor.
     */
    public ShowingCalendarEvents() {
        if(checkPIMSupport() == false) {
            exitMIDlet();
        }

        initializeComponents();
    }

    /**
     * Initializes components of midlet.
     */
    private void initializeComponents() {
        display = Display.getDisplay(this);

        // Create list of events
        eventListCtrl = new List("Events", List.IMPLICIT);
        cmdShowDetails = new Command("Details", Command.ITEM, 0);
        eventListCtrl.addCommand(cmdShowDetails);
        cmdExit = new Command("Exit", Command.EXIT, 0);
        eventListCtrl.addCommand(cmdExit);
        eventListCtrl.setCommandListener(this);

        addEventsToListCtrl();

        // Create event form
        eventForm = new Form("Event details");
        cmdBack = new Command("Back", Command.BACK, 0);
        eventForm.addCommand(cmdBack);
        eventForm.setCommandListener(this);
    }

    /**
     * Checks PIM support.
     * @return - true if PIM is supported, false otherwise.
     */
    private boolean checkPIMSupport() {
        String propValue = System.getProperty("microedition.pim.version");
        if(propValue != null) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * Adds events to list.
     */
    private void addEventsToListCtrl() {
        try {
```


(C) Copyright Nokia 2013. All rights reserved.

```
        // Create array for events
        eventsArray = new Vector();

        // Get list of events.
        EventList eventList = (EventList)PIM.getInstance().openPIMList(
                PIM.EVENT_LIST, PIM.READ_WRITE);

        Enumeration events = eventList.items();
        while(events.hasMoreElements() == true) {
            Event event = (Event)events.nextElement();
            // Add event to array of events
            eventsArray.addElement(event);
            // Add event's summary to list control
            String eventSummary = event.getString(Event.SUMMARY,
                    PIMItem.ATTR_NONE);
            eventListCtrl.append(eventSummary, null);
        }

        // Close list of events.
        eventList.close();

    } catch(PIMException pimExc) {
        // TODO: Handle error on working with PIM.
    } catch(Exception exc) {
        // TODO: handle error.
    }
}

/**
 * Shows form with details of selected event.
 */
private void showCurrentEvent() {
    try {
        // Get selected event.
        int eventIndex = eventListCtrl.getSelectedIndex();
        Event event = (Event)eventsArray.elementAt(eventIndex);
        if(event == null) {
            throw new Exception("No events were found.");
        }

        // Get list of events.
        EventList eventList = (EventList)PIM.getInstance().openPIMList(
                PIM.EVENT_LIST, PIM.READ_WRITE);

        // Delete from event form info about prevously shown event.
        eventForm.deleteAll();

        // Show summary of event
        if(eventList.isSupportedField(Event.SUMMARY) == true) {
            if(event.countValues(Event.SUMMARY) > 0) {
                String summary = event.getString(Event.SUMMARY,
                        PIMItem.ATTR_NONE);
                eventForm.append(new StringItem("Summary", summary));
            }
        }
        // Show start date
        if(eventList.isSupportedField(Event.START) == true) {
            if(event.countValues(Event.START) > 0) {
                String startDate = new Date(event.getDate(Event.START,
                        PIMItem.ATTR_NONE)).toString();
                eventForm.append(new StringItem("Start date", startDate));
            }
        }
        // Show end date
        if(eventList.isSupportedField(Event.END) == true) {
            if(event.countValues(Event.END) > 0) {
                String endDate = new Date(event.getDate(Event.END,
                        PIMItem.ATTR_NONE)).toString();
                eventForm.append(new StringItem("End date", endDate));
            }
        }
        // Show alarm
        if(eventList.isSupportedField(Event.ALARM) == true) {
            if(event.countValues(Event.ALARM) > 0) {
                int alarm = event.getInt(Event.ALARM,
                        PIMItem.ATTR_NONE);
                String alarmStr = Integer.toString(alarm);
                eventForm.append(new StringItem("Alarm", alarmStr));
            }
        }
        // Show note
        if(eventList.isSupportedField(Event.NOTE) == true) {
            if(event.countValues(Event.NOTE) > 0) {
                String note = event.getString(Event.NOTE,
                        PIMItem.ATTR_NONE);
                eventForm.append(new StringItem("Note", note));
            }
        }
        // Show location
        if(eventList.isSupportedField(Event.LOCATION) == true) {
            if(event.countValues(Event.LOCATION) > 0) {
                String note = event.getString(Event.LOCATION,
                        PIMItem.ATTR_NONE);
                eventForm.append(new StringItem("Location", note));
            }
        }
        // Show revision
        if(eventList.isSupportedField(Event.REVISION) == true) {
            if(event.countValues(Event.REVISION) > 0) {
                Date revisionDate = new Date(event.getDate(Event.REVISION,
                        PIMItem.ATTR_NONE));
                String revision = revisionDate.toString();
                eventForm.append(new StringItem("Revision", revision));
            }
        }

        // Close list of events.
        eventList.close();

        // Show form with details of selected event.
        display.setCurrent(eventForm);

    } catch(PIMException pimExc) {
        // TODO: Handle error on working with PIM.
        showAlert("PIMException", pimExc.getMessage());
    } catch(Exception exc) {
        // TODO: catch all other errors.
        showAlert("Exception", exc.getMessage());
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

```
    /**
     * Shows alert with specified title and text.
     * @param title - Title of alert.
     * @param message - text of alert.
     */
    private void showAlert(String title, String message) {
        Alert alert = new Alert(title);
        alert.setString(message);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
    }

    /**
     * From MIDlet.
     * Signals the MIDlet that it has entered the Active state.
     */
    public void startApp() {
        display.setCurrent(eventListCtrl);
    }

    /**
     * From MIDlet.
     * Signals the MIDlet to enter the Paused state.
     */
    public void pauseApp() {
        // No implementation required.
    }

    /**
     * From MIDlet.
     * Signals the MIDlet to terminate and enter the Destroyed state.
     */
    public void destroyApp(boolean unconditional) {
        // No implementation required.
    }

    /**
     * Performs exit from midlet.
     */
    private void exitMIDlet() {
        notifyDestroyed();
    }

    /**
     * From CommandListener.
     * Indicates that a command event has occurred on Displayable displayable.
     * @param command - a Command object identifying the command.
     * @param displayable - the Displayable on which this event has occurred.
     */
    public void commandAction(Command command, Displayable displayable) {
        // Handles "show details" command on selected event.
        if(command == cmdShowDetails) {
            showCurrentEvent();
        }
        // Handles "exit" command.
        if(command == cmdExit) {
            exitMIDlet();
        }
        // Handles "back" command.
        if(command == cmdBack) {
            display.setCurrent(eventListCtrl);
        }
    }
}
```

Postconditions

This code snippet represents a MIDlet with a list control which contains summaries of events from the PIM list of calendar events.

By choosing an event from the list and pressing the "Details" menu command, the user can see the fields of the selected event.

Supplementary material

Executables and source files are available at Media:Showing calendar events.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.