Localising text files in Java ME

Overview

This code snippet is one of the series of snippets that demonstrate how to implement multi-language applications in Java ME. There are several techniques for localizing a MIDlet:

- Using application attributes
- Using text files
- Using resource bundles

This snippet describes the second technique; using text files.

In the technique presented in this snippet, localized strings are stored in text files. This allows adding languages easily since no coding is required. However, the developer needs to write his or her own stream parser in order to process the text files. Also, the string resources need to be loaded and held in an internal data structure as long as they are used. Loading strings also increases the startup time. Also note that text files address only string resources.

Text file: strings-en.txt

```
title|Localization example
exit|Exit
localeLbl|Locale
textLbl|Text
text|Here's some text.
```

Text file: strings-fr.txt

```
exit|Quitter
localeLbl|Locale
textLbl|Texte
text|Voici du texte.
```

Note: In the French text file (`strings-fr.txt`), the title is omitted to demonstrate the usage of default values when retrieving the resources. See the `localize()` method for implementation details.

Source

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Hashtable;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.midlet.MIDlet;
```

```
public class L10nMIDlet extends MIDlet implements CommandListener {
    private Form mainForm;
    private Command exitCommand;
    private Hashtable strings;
```


(C) Copyright Nokia 2014. All rights reserved.

```
    private String locale;
    /** The character used for separating the key from the value. */
    private static final char SEPARATOR_CHAR = '|';

    /**
     * Constructor. Constructs the object and initializes displayables.
     */
    public L10nMIDlet() {
        // Obtain the system locale
        locale = System.getProperty("microedition.locale");
        String filename = "strings-" + locale + ".txt";
        try {
            loadStrings(filename);
        } catch (IOException ex) {
            // TODO: Exception handling
        } catch (Exception ex) {
            // TODO: Exception handling
        }

        initUI();
    }

    /**
     * Stores the strings from the text file into a hash table as key-value
     * pairs.
     * @param filename
     * @throws java.lang.Exception
     */
    private void loadStrings(final String filename) throws Exception {
        strings = new Hashtable(50);
        InputStreamReader reader = new InputStreamReader(
            getClass().getResourceAsStream(filename));
        String line = null;
        // Read a single line from the file. null represents the EOF.
        while ((line = readLine(reader)) != null) {
            // Empty lines are ignored
            if (line.trim().equals("")) {
                continue;
            }
            // Separate the key from the value and put the strings into the
            // hash table
            int separatorPos = line.indexOf(SEPARATOR_CHAR);
            if (separatorPos == -1) {
                throw new Exception("Separator character not found.");
            }
            String key = line.substring(0, separatorPos);
            String value = line.substring(separatorPos + 1);
            strings.put(key, value);
        }
        reader.close();
    }

    /**
     * Reads a single line using the specified reader.
     * @throws java.io.IOException if an exception occurs when reading the line
     */
    private String readLine(InputStreamReader reader) throws IOException {
```


(C) Copyright Nokia 2014. All rights reserved.

```
        // Test whether the end of file has been reached. If so, return null.
        int readChar = reader.read();
        if (readChar == -1) {
            return null;
        }
        StringBuffer string = new StringBuffer("");
        // Read until end of file or new line
        while (readChar != -1 && readChar != '\n') {
            // Append the read character to the string. Some operating systems
            // such as Microsoft Windows prepend newline character ('\n') with
            // carriage return ('\r'). This is part of the newline character and
            // therefore an exception that should not be appended to the string.
            if (readChar != '\r') {
                string.append((char)readChar);
            }
            // Read the next character
            readChar = reader.read();
        }
        return string.toString();
    }

    public void initUI() {
        // Initialize the form
        mainForm = new Form(localize("title", "Localization example"));

        // Display the locale
        StringItem localeItem = new StringItem(localize("localeLbl", "Locale"),
            locale);
        mainForm.append(localeItem);

        // Display a localized text string
        StringItem textItem = new StringItem(localize("textLbl", "Text"),
            localize("text", "Here's some text."));
        mainForm.append(textItem);

        // Create the exit command
        exitCommand = new Command(localize("exit", "Exit"), Command.EXIT, 1);
        mainForm.addCommand(exitCommand);
        mainForm.setCommandListener(this);
    }

    /**
     * Localizes a string.
     * @param key the key used for identifying a specific value in the text
     *      file
     * @param def the default value in case "key" is not found from the text
     *      file
     * @return the message localized; "def" if the key is not found
     */
    private String localize(String key, String def) {
        // Obtain the localized string from
        String value = (String)strings.get(key);
        if (value == null) {
            value = def;
        }
        return value;
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }

    /**
     * From MIDlet.
     * Called when the MIDlet is started.
     */
    public void startApp() {
        // The initial display is the first form
        Display.getDisplay(this).setCurrent(mainForm);
    }

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

    /**
     * From MIDlet.
     * Called to signal the MIDlet to terminate.
     * @param unconditional whether the MIDlet has to be unconditionally
     * terminated
     */
    public void destroyApp(boolean unconditional) {
        // No implementation required
    }

    /**
     * From CommandListener.
     * Called by the system to indicate that a command has been invoked on a
     * particular displayable.
     * @param command the command that was invoked
     * @param displayable the displayable where the command was invoked
     */
    public void commandAction(Command command, Displayable displayable) {
        if (command == exitCommand) {
            // Exit the MIDlet
            notifyDestroyed();
        }
    }
}
```

Postconditions

A MIDlet is localized using text files.

See also

- Reading a text file line by line in Java ME
- Localising app attributes in Java ME
- Localising resource bundles in Java ME

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.