Multilingual translation of Cordova applications

Overview

You can use the IBM MobileFirst Platform Foundation framework to add multilingual translation of Cordova applications into other languages.
Items that can be translated are application strings and system messages.

Jump to:

- Translating application strings
- Translating system messages
- Multilanguage translation
- Detecting the device locale and language
- Sample application

Translating application strings

Strings that are destined to be translated are stored in a `JSON` object called "Messages". You can find it in the index.js file of the Cordova application: **[cordova-project-root-directory]/www/js/index.js**.

JSON object structure example

```
var Messages = {
   headerText: "Default header",
   actionsLabel: "Default action label",
   sampleText: "Default sample text",
};
```

Strings stored in the Messages `JSON` object can be referenced in two ways in the application logic:

As a JavaScript object property:

```
Messages.headerText
```

As an ID of an HTML element with `class="translate"`:

```
<h1 id="headerText" class="translate"></h1>
```

Translating system messages

It is also possible to translate the system messages that the application displays, for example "Internet connection is not available" or "Invalid username or password". System messages are stored in the `WL.ClientMessages` object.

You can find a full list of system messages in the `messages.json` file, located inside the generated project.

- Android: `[Cordova-project]\platforms\android\assets\www\plugins\cordova-plugin-mfp\worklight\messages`
- iOS, Windows: `[Cordova-project]\platforms\[ios or windows]\www\plugins\cordova-`

`plugin-mfp\worklight\messages`

To translate a system message, override it in the application code.

```
WL.ClienMessages.loading = "Application HelloWorld is loading... please wait.";
```

Note: Override system messages at a global JavaScript level because some parts of the code are executed only after the application has successfully initialized.

Multilanguage translation

Using JavaScript, you can implement multilanguage translation for your application.
The below steps explain the implementation of this tutorial's sample application.

1. Set up the default application strings in the `index.js` file.

   ```
   var Messages = {
      headerText: "Default header",
      actionsLabel: "Default action label",
      sampleText: "Default sample text",
      englishLanguage: "English",
      frenchLanguage: "French",
      russianLanguage: "Russian",
      hebrewLanguage: "Hebrew"
   };
   ```

2. Override specific strings when required.

   ```
   function setFrench(){
      Messages.headerText = "Traduction";
      Messages.actionsLabel = "Sélectionnez une langue:";
      Messages.sampleText = "Ceci est un exemple de texte en français.";
   }
   ```

3. Update the GUI components with the new strings. You can perform more tasks, such as setting the text direction for right-to-left languages such as Hebrew or Arabic. Each time that an element is updated, it is updated with different strings according to the active language.

```
function languageChanged(lang) {
    if (typeof(lang)!="string")
        lang = $("#languages").val();

    switch (lang) {
        case "english":
            setEnglish();
            break;
        case "french":
            setFrench();
            break;
        case "russian":
            setRussian();
            break;
        case "hebrew":
            setHebrew();
            break;
    }

    if ($("#languages").val()=="hebrew")
        $("#wrapper").css({direction: 'rtl'});
    else
        $("#wrapper").css({direction: 'ltr'});

    $("#sampleText").html(Messages.sampleText);
    $("#headerText").html(Messages.headerText);
    $("#actionsLabel").html(Messages.actionsLabel);
}
```

Detecting the device locale and language

It is possible to detect the locale and the language of the device using the Cordova's globalization plug-in: `cordova-plugin-globalization`.
The globalization plug-in is auto-installed when adding a platform to the Cordova application.

Use the `navigator.globalization.getLocaleName` and `navigator.globalization.getPreferredLanguage` functions to detect the locale and language respectively.

```
navigator.globalization.getLocaleName(
    function (localeValue) {
        WL.Logger.debug(">> Detected locale: " + localeValue);

        ...
        ...
        ...
    },
    function() {
        WL.Logger.debug(">> Unable to detect locale.");
    }
);

navigator.globalization.getPreferredLanguage(
    function (langValue) {
        lang = langValue.value;
        WL.Logger.debug(">> Detected language: " + lang);
    },
    function() {
        WL.Logger.debug(">> Unable to detect language.");
    }
);
```

The result can then be seen in the device log, for example from Android's LogCat:

```
12-22 15:43:41.370 2093-2108/com.mfp.translation D/NONE: >> Detected language: en-US
12-22 15:43:41.380 2093-2108/com.mfp.translation D/NONE: before: initOptions.onSuccess
```

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/Translation) the Cordova project.

Sample usage

1. From the command line, navigate to the Cordova project.
2. Add a platform using the `cordova platform add` command.
3. Prepare and run the Cordova application using `cordova prepare` followed by `cordova run`.

> Tip: you can inspect Android's LogCat from Android Studio's LogCat console while the application is running.