IBM Worklight

IBM Worklight V5.0.6

Globalization white paper

11 April 2013

About IBM®

See http://www.ibm.com/ibm/us/en/.

Contents

Figures and Tables

About this document

This document is intended for users who want to develop globalized hybrid applications. With this document, users can learn about globalization in JavaScript™ frameworks and IBM® Worklight®, and about globalizing web services and push notifications.

1 Introduction

The increased use of mobile devices has dramatically changed our lives. Thousands of applications are developed and uploaded to application stores every day, and they must support multiple languages if they are to be used globally. The IBM Worklight platform provides rich capabilities for you to develop globalized hybrid applications.

2 Globalization in JavaScript frameworks

2.1 Dojo globalization framework

The following example application demonstrates how to use the Dojo Mobile JavaScript API to construct a globalized application with a native look and feel. Dojo Mobile provides globalization functions for detecting locale, loading and accessing resource bundles, and simple formatting utilities for culture-sensitive display. *Figure 2-1* and *Figure 2-2* show pages of the application that display the resource bundles loaded and the string format that is determined by the user preferences on the device.

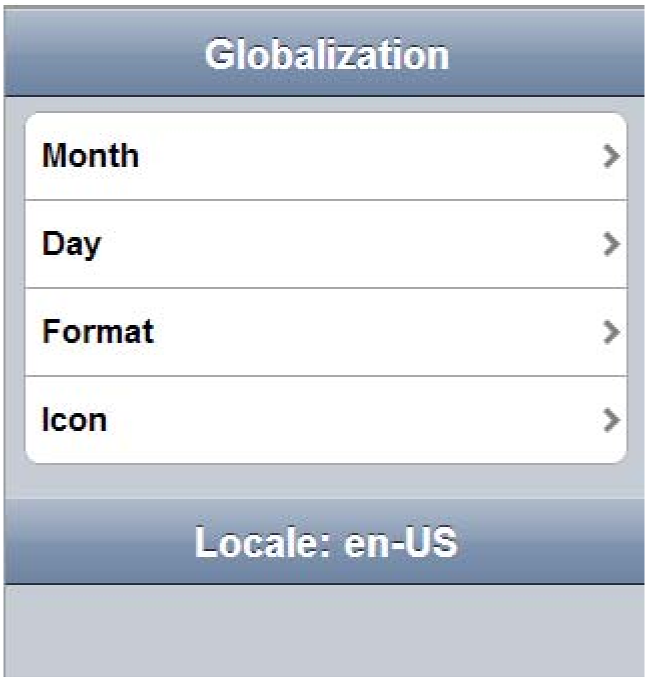
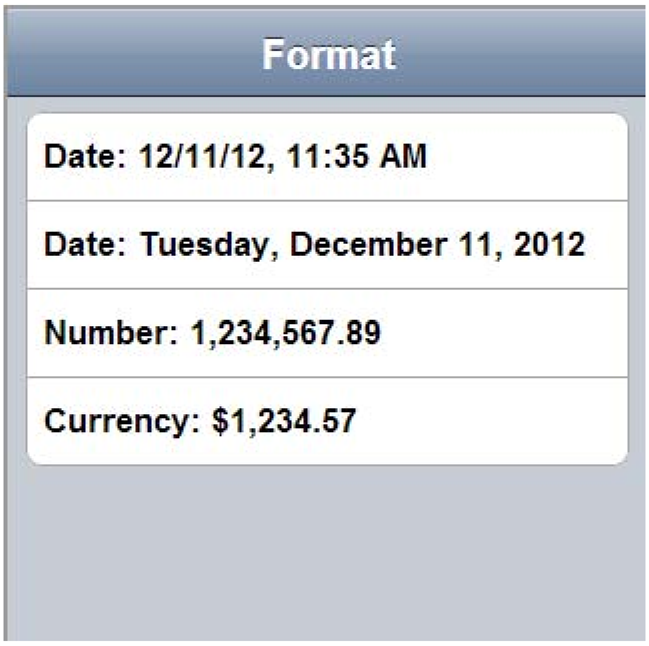


Figure 2-1: Dojo Globalization application



Date: 12/11/12, 11:35 AM

Date: Tuesday, December 11, 2012

Number: 1,234,567.89

Currency: $1,234.57

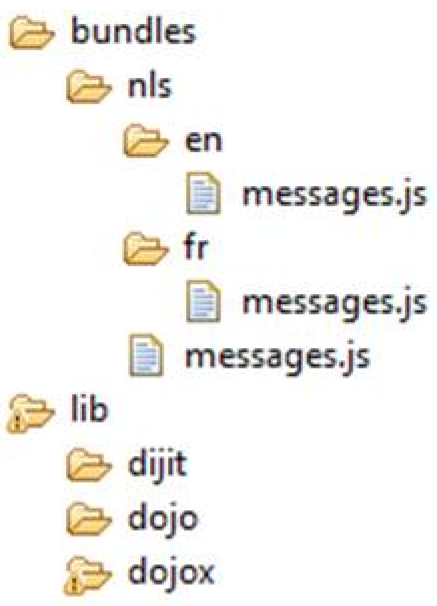
Figure 2-2: Dojo cultural formatting

In the example, the Dojo library is loaded as shown in **Listing 1**. The required modules must be loaded before you can use the Dojo globalization API.

Listing 1: Including Dojo Mobile

```
<script type="text/javascript">
  var dojoConfig = {
    parseOnLoad: false,
    packages: [{
      name: "resource",
      location: "../../bundles"
    }]
  };
</script>
<script type="text/javascript" src="libs/dojo/dojo.js"></script>
```

Figure 2-3 shows how to load Dojo resource bundles by defining a package that maps the location of resource files within your hybrid application to a package name.



```
bundles
    nls
        en
            messages.js
        fr
            messages.js
        messages.js
lib
    dijit
    dojo
    dojox
```

Figure 2-3: Dojo resource bundle structure

In the example, the language resource bundles are part of the application package, and are stored on the client-side instead of being supplied dynamically from the server or inserted directly into the HTML markup. Storing the language resource bundles on the client-side enables the application to be used offline. The resource files are encoded as JSON files. **Listing 2** shows the resources files that are loaded by the `dojo/i18n` plug-in using Asynchronous Module Definition (AMD).

Listing 2: Loading modules and resource files with the Dojo Mobile resource bundle API

```
require(
  [
  "dojo/domReady",                // Make sure DOM are ready
  "dojo/i18n!resource/nls/messages", // Load our resource bundle
  "dojox/mobile/parser",          // This mobile app uses declarative
programming
  "dojox/mobile",                 // This is a mobile app
  "dojox/mobile/i18n",            // Load resources bundle
declaratively
  "dojox/mobile/deviceTheme",     // Automatically Applying Mobile
Device Themes Using CSS
  "dojox/mobile/compat",          // This mobile app supports running
on desktop browsers
  ],
  function(ready, messages, parser, mobile, mi18n) {
    ready( function() {
      // demonstrates how to load resources declaratively
      // dojox.mobile.i18n.load() must be called before
dojox.mobile.parser.parse()
      mi18n.load("resource", "messages");
      parser.parse();
    });
  }
);
```

Note: The `dojo.18n.getLocalization` API is deprecated. Use `dojox/mobile/i18n` to load resources declaratively. The `dojox/mobile/i18n load()` method treats text in all mobile widgets as resource keys, and automatically replaces those keys with the actual resources. If you want to explicitly control how these resources are used, they can be loaded programmatically. **Listing 3** shows how to use an argument such as "resource" to retrieve loaded resources. **Listing 4** shows the Dojo cultural formatting functions. **Listing 5** shows the code to generate the pages as simple HTML markup.

Listing 3: Explicitly using the loaded resources

```
require(
  [
  "dojo/domReady",             // Make sure DOM are ready
  "dojo/i18n!resource/nls/messages", // Load our resource bundle
  "dijit/registry",            // For registry.byId
  "dojox/mobile/parser",       // This mobile app uses declarative
programming
  "dojox/mobile",              // This is a mobile app
  "dojox/mobile/deviceTheme", // Automatically Applying Mobile Device
Themes Using CSS
  "dojox/mobile/compat",       // This mobile app supports running on
desktop browsers
  ],
  function(ready, messages, parser, registry) {
    ready( function() {
      parser.parse();
      registry.byId("globalization_heading").set("label",
messages["msg_globalization"]);
      registry.byId("months_choice").set("label",
messages["msg_months"]);
      registry.byId("days_choice").set("label",
messages["msg_days"]);
      registry.byId("formats_choice").set("label",
messages["msg_formats"]);
      // get locale by dojo
      var footer_msg = bundle["msg_footer"] + dojo.locale;
      registry.byId("globalization_footer").set("label", footer_msg);
    });
  }
);
```

Listing 4: Dojo cultural formatting

```
function getFormats(){
  var formatsView = dojo.byId("formats");
  require(
    [
    "dojox/mobile/RoundRectList",
    "dojox/mobile/ListItem",
    "dojo/date/locale",
    "dojo/number",
    "dojo/currency"
    ],
    function(RoundRectList, ListItem, localeDate, localeNumber,
localeCurrency){
      var formatsList = new RoundRectList({id:
"formats_list"}).placeAt(formatsView);
      // get locale by dojo
      var myLocale = dojo.locale;
      // format locale date by dojo/date/locale
      var date = localeDate.format(new Date(), {locale: myLocale});
      var formattedDate = new ListItem({label: "Date: " + date});
      formatsList.addChild(formattedDate);
      // format with parameter
      date = localeDate.format(new Date(), {selector: 'date',
formatLength: 'full'});
      formattedDate = new ListItem({label: "Date: " + date});
      formatsList.addChild(formattedDate);
      // format number
      var number = localeNumber.format(1234567.89);
      var formattedNumber = new ListItem({label: "Number: " +
number});
      formatsList.addChild(formattedNumber);
      // format currency
      var currency = localeCurrency.format(1234.567, {currency:
"USD"});
      var formattedCurrency = new ListItem({label: "Currency: " +
currency});
      formatsList.addChild(formattedCurrency);
    }
  );
};
```

Listing 5: Globalization application Views

```
<!--  Main page  -->
<div id="globalization" data-dojo-type="dojox.mobile.View"
selected="true">
  <h1 id="globalization_heading" data-dojo-
type="dojox.mobile.Heading" label="msg_globalization"></h1>
  <ul data-dojo-type="dojox.mobile.RoundRectList">
    <li id="months_choice" data-dojo-type="dojox.mobile.ListItem"
moveTo="months" callback="getMonths" label="msg_months"></li>
    <li id="days_choice" data-dojo-type="dojox.mobile.ListItem"
moveTo="days" callback="getDays" label="msg_days"></li>
    <li id="formats_choice" data-dojo-type="dojox.mobile.ListItem"
moveTo="formats" callback="getFormats" label="msg_formats"></li>
    <li id="icon_choice" data-dojo-type="dojox.mobile.ListItem"
label="msg_icon"></li>
  </ul>
  <h1 id="globalization_footer" data-dojo-type="dojox.mobile.Heading"
fixed="bottom" label="msg_footer" ></h1>
</div>

<!-- The "Icon" sub-page -->
<div id="icons" data-dojo-type="dojox.mobile.View">
  <h1 id="icon_heading" data-dojo-type="dojox.mobile.Heading"
moveTo="globalization" label="msg_icon" back="msg_previous"></h1>
</div>
```

```
<!-- The "Months" sub-page -->
<div id="months" data-dojo-type="dojox.mobile.View">
  <h1 id="months_heading" data-dojo-type="dojox.mobile.Heading"
moveTo="globalization" label="msg_months" back="msg_previous"></h1>
</div>

<!-- The "Days" sub-page -->
<div id="days" data-dojo-type="dojox.mobile.View">
  <h1 id="days_heading" data-dojo-type="dojox.mobile.Heading"
moveTo="globalization" label="msg_days" back="msg_previous"></h1>
</div>

<!-- The "Formats" sub-page -->
<div id="formats" data-dojo-type="dojox.mobile.View">
  <h1 id="formats_heading" data-dojo-type="dojox.mobile.Heading"
moveTo="globalization" label="msg_formats" back="msg_previous"></h1>
</div>
```

2.2 jQuery Mobile globalization plug-in

There are no official jQuery globalization bundles. In this white paper, the `jquery.i18n.properties-1.0.9.js` jQuery globalization plug-in is being used to demonstrate jQuery globalization functions. The `jquery.i18n.properties-1.0.9.js` jQuery globalization plug-in can be downloaded from http://code.google.com/p/jquery-i18n-properties/.

The example application does not show the jQuery globalization string format feature because there is no official globalization string formatting plug-in for jQuery frameworks.

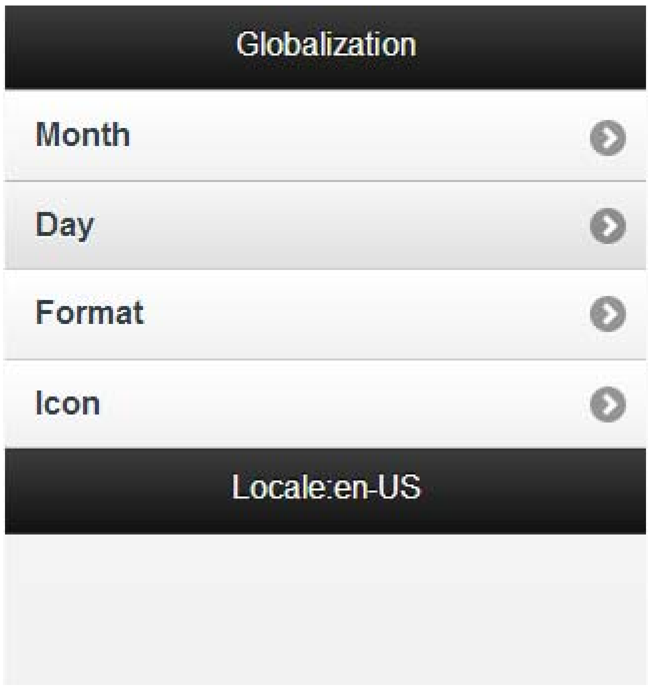


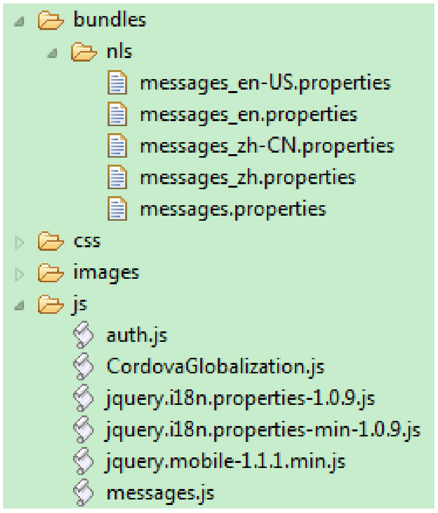
Figure 2-4: jQuery Globalization application

Listing 6 shows the scripts for loading Cordova, jQuery mobile, and the `jquery.i18n.properties-1.0.9.js` jQuery globalization plug-in.

Listing 6: Load Cordova, jQuery mobile, and jQuery globalization plug-in

```
<script
  type="text/javascript"
  src="js/CordovaGlobalization.js">
</script>
<script
  type="text/javascript"
  src="js/messages.js">
</script>
<script
  type="text/javascript"
  src="js/jquery.mobile-1.1.1.min.js">
</script>
<script
  type="text/javascript"
  src="js/jquery.i18n.properties-min-1.0.9.js">
</script>
```

The resource bundle structures in jQuery and Dojo are different. Dojo resource files have the same file name but are located in separate folders corresponding to the locale name. jQuery resource files are located in one folder but the file names include the locale information. *Figure 2-5* shows the structure of the jQuery resource files.



```
bundles
  nls
    messages_en-US.properties
    messages_en.properties
    messages_zh-CN.properties
    messages_zh.properties
    messages.properties
css
images
js
  auth.js
  CordovaGlobalization.js
  jquery.i18n.properties-1.0.9.js
  jquery.i18n.properties-min-1.0.9.js
  jquery.mobile-1.1.1.min.js
  messages.js
```

Figure 2-5: jQuery resource bundle structure

Listing 7 shows the jQuery scripts to initialize the globalization plug-in.

Listing 7: Load and use resource by jQuery globalization plug-in

```
function doGlobalization(){
  $.i18n.properties({
    name: 'messages',
    path: 'bundles/nls/',
    mode: 'both',
    // language: 'zh',
    callback: function(){
      // Main page
      $('#globalization_heading').empty().
        append($.i18n.prop('msg_globalization'));
      $('#msg_months').empty().append($.i18n.prop('msg_months'));
      $('#msg_days').empty().append($.i18n.prop('msg_days'));
      $('#msg_formats').empty().append($.i18n.prop('msg_formats'));
      $('#msg_icon').empty().append($.i18n.prop('msg_icon'));
      // Sub page heading
      $('#icon_heading').empty().append($.i18n.prop('msg_icon'));
      $('#months_heading').empty().append($.i18n.prop('msg_months'));
      $('#days_heading').empty().append($.i18n.prop('msg_days'));
      $('#formats_heading').empty().append($.i18n.prop('msg_formats'));
      $('#words_heading').empty().append($.i18n.prop('msg_words'));
      //Back buttons
      var items = $('a[data-rel="back"]');
      $.each(items, function(i){
        $(items[i]).empty().append($.i18n.prop('msg_previous'));
      });
      //Show locale by jQuery i18n plug-in
      $('#globalization_footer').empty().
        append($.i18n.prop('msg_footer') + $.i18n.browserLang());
    }
  });
};
```

2.3 Sencha Touch globalization bundle

There are no official Sencha Touch globalization bundles. In this white paper, the `Ext.i18n.bundle-touch` Sencha Touch globalization plug-in is being used to demonstrate globalization functions. The `Ext.i18n.bundle-touch` globalization plug-in can be downloaded from http://gaver.dyndns.org/elmasse/site/index.php/download-menu/9-sencha-touch/21-exti18nbundle-touch-downloads.

The example application does not show the Sencha Touch globalization string format feature because there is no official globalization string formatting plug-in for Sencha frameworks.

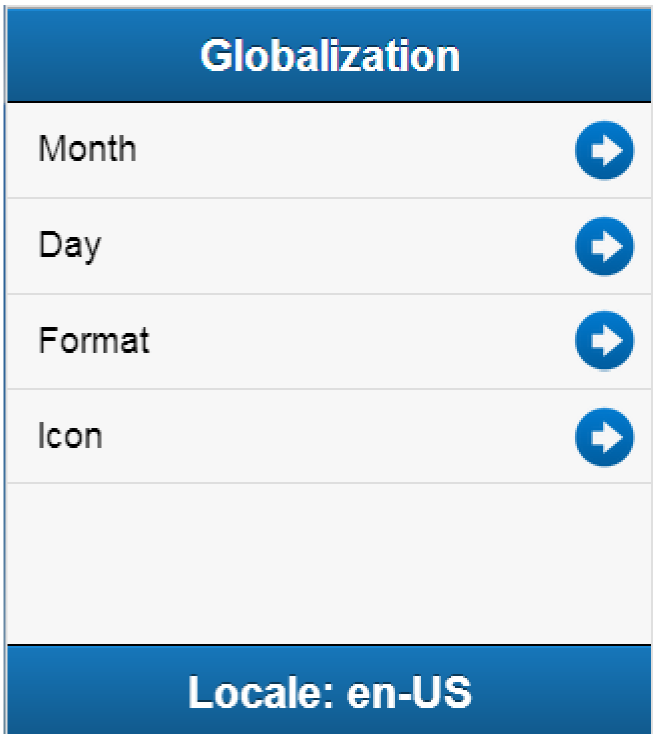


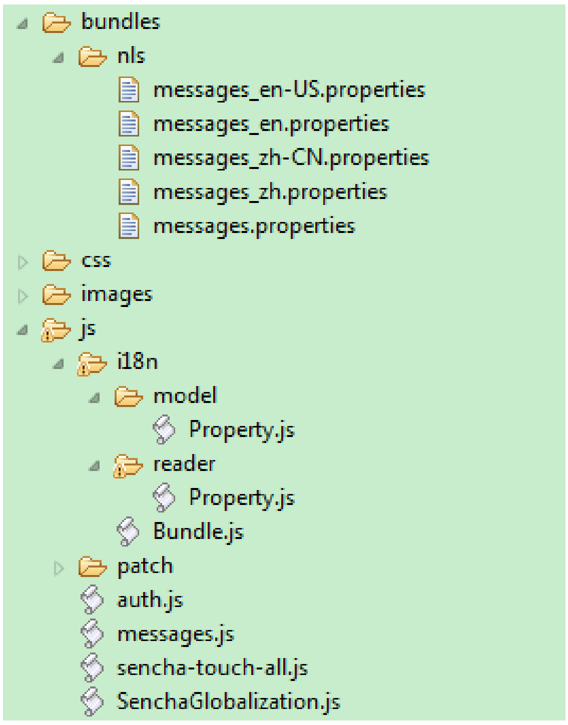
Figure 2-6: Sencha Touch Globalization application

Listing 8 shows the scripts for loading Sencha Touch and the `Ext.i18n.bundle-touch` globalization plug-in.

Listing 8: Load Sencha Touch and globalization plug-in

```
<script src="js/sencha-touch-all.js"></script>
<script>
  Ext.Loader.setConfig({
    enabled: true,
    paths: {
      'Ext.i18n': 'js/i18n',
      'patch': 'js/patch'
    }
  });
</script>
<script src="js/SenchaGlobalization.js"></script>
<script src="js/messages.js"></script>
<script src="js/auth.js"></script>
```

Figure 2-7 shows the structure of the Sencha Touch resource files.



```
bundles
  nls
    messages_en-US.properties
    messages_en.properties
    messages_zh-CN.properties
    messages_zh.properties
    messages.properties
css
images
js
  i18n
    model
      Property.js
    reader
      Property.js
    Bundle.js
  patch
  auth.js
  messages.js
  sencha-touch-all.js
  SenchaGlobalization.js
```

Figure 2-7: Sencha Touch resource bundle structure

Sencha Touch also provides a convenient API to retrieve the message in the resource bundle and set the value to the UI component.

Listing 9: Load and use resource by Sencha Touch globalization plug-in

```
function loadResource(){
  Ext.require('Ext.i18n.Bundle', function(){
    Ext.i18n.appBundle = Ext.create('Ext.i18n.Bundle', {
      bundle: 'messages',
      path: 'bundles/nls',
      noCache: true
    });
  });
  Ext.application({
    name: "Sencha Touch Globalization",
    launch: function(){
      Ext.i18n.appBundle.onReady(function(){doGlobalization();});
    }
  });
};

function doGlobalization(){
  // global header
  var globalHeader = Ext.create('Ext.Toolbar', {
    docked: 'top',
    xtype: 'toolbar',
    title: '<div width="100px">' +
      Ext.i18n.appBundle.getMsg('msg_globalization') + '</div>'
  });
  // show locale by Ext.i18n.Bundle
  var globalFooter = Ext.create('Ext.Toolbar', {
    docked: 'bottom',
    xtype: 'toolbar',
    title: '<div width="100px">' +
      Ext.i18n.appBundle.getMsg("msg_footer") +
      Ext.i18n.appBundle.language + '</div>'
  });
  // main list data model
  Ext.define('mainListModel', {
    extend: 'Ext.data.Model',
    config: {fields: ['index', 'type']}
  });
```

```
// main list data store
var mainListStore = Ext.create('Ext.data.Store', {
  model: 'mainListModel',
  sorters: 'index',
  proxy: {
    type: 'localstorage',
    id: 'mainListStore'
  },
  data: [
    {
      index: '1',
      type: Ext.i18n.appBundle.getMsg('msg_months')
    },
    {
      index: '2',
      type: Ext.i18n.appBundle.getMsg('msg_days')
    },
    {
      index: '3',
      type: Ext.i18n.appBundle.getMsg('msg_formats')
    },
    {
      index: '4',
      type: Ext.i18n.appBundle.getMsg('msg_words')
    },
    {
      index: '5',
      type: Ext.i18n.appBundle.getMsg('msg_icon')
    }
  ]
});
// main list view
var mainList = Ext.create('Ext.List', {
  itemTpl: '{type}',
  store: mainListStore,
  onItemDisclosure: function(record, btn, index){
    showSecondContainer(record, btn, index);
  }
});
```

```
}
```

In summary, Dojo, jQuery, and Sencha Touch each provide globalization functions that are based on resource bundles and resource files, and they can switch to different resource bundles based on current locale information. In addition, Dojo also provides string and date format utilities that are based on user locale information.

3 Globalization in IBM Worklight

Applications can be translated into other languages within the IBM Worklight framework. The following resources can be translated:

- Application strings
- System messages

The IBM Worklight Platform automatically translates application strings according to a designated file. Multi-language translation is implemented by using JavaScript.

3.1 Cordova Globalization API

The Cordova globalization API provides enhanced globalization capabilities that mirror existing JavaScript globalization functions, where possible, without duplicating functions already present in JavaScript. The emphasis of the Cordova globalization API is on parsing and formatting culturally sensitive data. The Cordova API uses native functions in the underlying operating system, where possible, rather than re-creating these functions in JavaScript. *Table 3-1* summarizes the Cordova globalization API functions provided.

Function Name	Purpose
getLocaleName	Client current locale setting on device
dateToString	Date formatted as string using client preferences
stringToDate	String parsed as date using the client preferences
getDatePattern	Pattern string for formatting and parsing dates
getDateNames	Names of months or days of week
isDayLightSavingsTime	Is daylight savings time in effect for a specified date
getFirstDayOfWeek	First day of week
numberToString	Number formatted as string using the preferences
stringToNumber	String parsed as number using client preferences
getNumberPattern	Pattern string for formatting and parsing numbers
getCurrencyPattern	Pattern string for formatting and parsing currencies

Table 3-1: Cordova globalization API summary

The Cordova globalization API is an independent globalization framework, which can be integrated with any JavaScript libraries to provide globalization functions. The Cordova globalization API is different from other globalization libraries. The Cordova globalization API does not provide a parameter to indicate a locale. The set of supported locales is determined by the device and its SDK and not by Cordova.

The Cordova globalization API uses the client locale setting and any default values that are overridden. This design greatly simplifies the use of the globalization API while still providing robust support. It is important to note that, although the set of interfaces remains constant across the devices that Cordova supports, the results can vary across the devices.

The Cordova framework does not provide access to graphical widgets that are present in device SDKs. The Cordova framework is used in concert with other JavaScript widget libraries, such as Dojo, to build complete mobile applications. The Cordova globalization API is interoperable with Dojo Mobile, jQuery Mobile, and Sencha Touch. It is an asynchronous implementation to prevent blocking JavaScript execution in user interface code. The following listings and figures show the Cordova globalization API. Dojo is used to demonstrate the user interface.

Listing 10: Using the Cordova globalization API

```
function onDeviceReady(){
  g11n = window.plugins.globalization;
}
```

The code to generate the names of the months, days of the week, and format the current date is shown in **Listing 11**, **Listing 12**, and **Listing 13**.

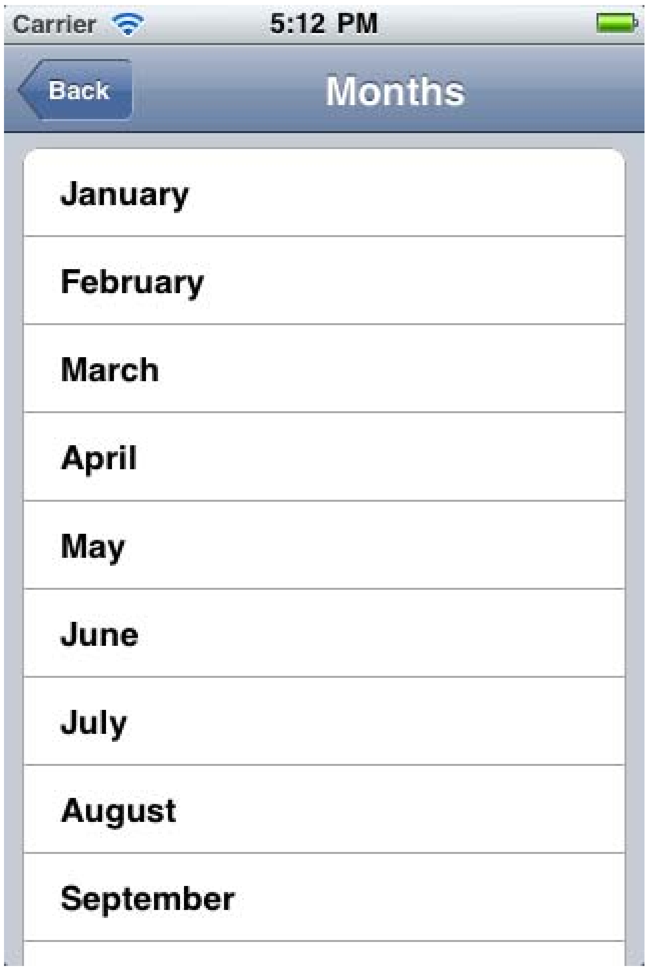
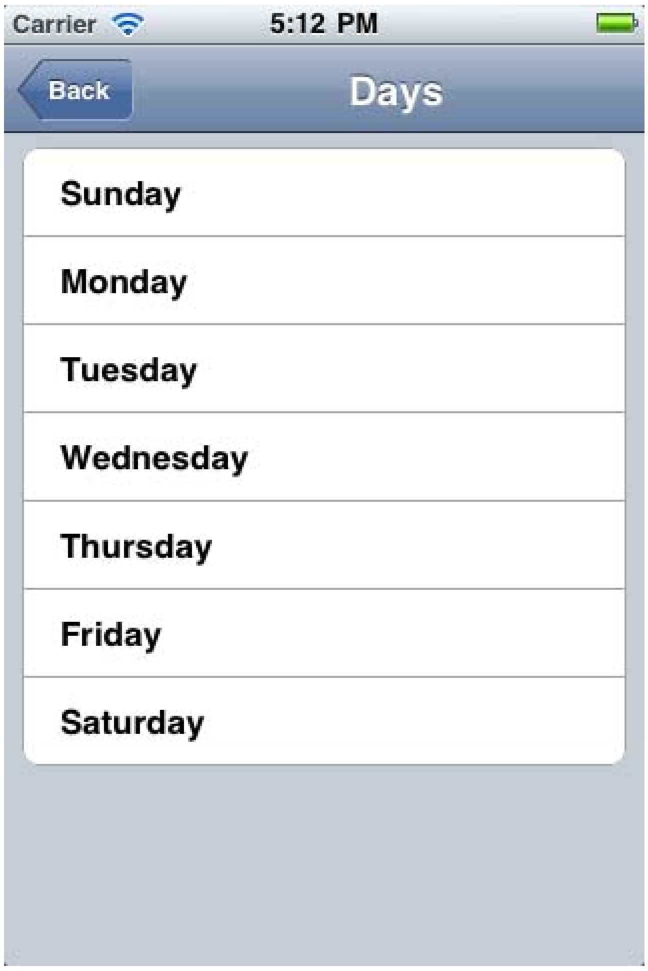


Figure 3-1: Using Cordova to show locale-based months

Listing 11: Month names

```
function getMonths(){
  g11n.getDateNames(function(data){
    var items = data.value;
    var monthsView = document.getElementById('monthsView');
    for (var i = 0; i < items.length; i++) {
      monthsView.append('<li>' + items[i] + '</li>');
    }
  },
  function(code){
    alert("Error: " + code);
  });
};
```

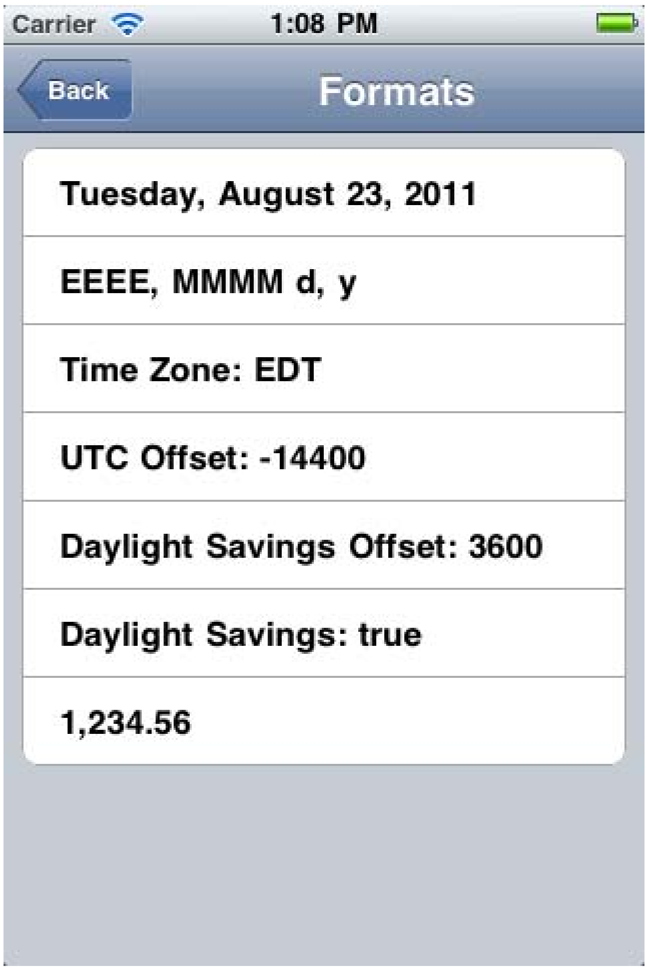


Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

Figure 3-2: Using Cordova to show the days of week

Listing 12: Days of the week

```
function getDays(){
  g11n.getDateNames(function(data){
    var items = data.value;
    var daysView = document.getElementById('daysView');
    for (var i = 0; i < items.length; i++) {
      daysView.append('<li>' + items[i] + '</li>');
    }
  },
  function(code){
    alert("Error: " + code);
  },
  {
    item: "days"
  });
}
```



Tuesday, August 23, 2011

EEEE, MMMM d, y

Time Zone: EDT

UTC Offset: -14400

Daylight Savings Offset: 3600

Daylight Savings: true

1,234.56

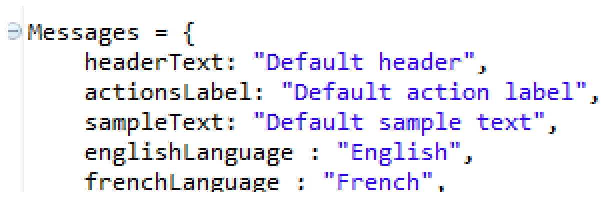
Figure 3-3: Using Cordova for cultural formatting

Listing 13: Formatting current date

```
function getFormats(){
  var formatsView = document.getElementById('formatsView');
  g11n.dateToString(
    new Date(),
    function(date){
      formatsView.append('<li>' + date.value + '</li>');
    },
    function(code){alert("Error: " + code);},
    {selector: "date", formatLength: "full"}
  );
  g11n.getDatePattern(
    function(date){
      formatsView.append('<li>' + date.pattern + '</li>');
      var timeZone = date.timezone;
      formatsView.append('<li>' + timeZone + '</li>');
      var offset = date.utc_offset;
      formatsView.append('<li>' + offset + '</li>');
      var dstoffset = date.dst_offset;
      formatsView.append('<li>' + dstoffset + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {selector: "date", formatLength: "full"}
  );
  g11n.isDayLightSavingsTime(
    new Date(),
    function(date){
      var dst = date.dst;
      formatsView.append('<li>' + dst + '</li>');
    },
    function(code){
     alert("Error: " + code);
    }
  );
  g11n.numberToString(
    1234.56,
    function(number){
      formatsView.append('<li>' + number.value + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {type: "decimal"}
  );
}
```

3.2 Enabling translation of application strings

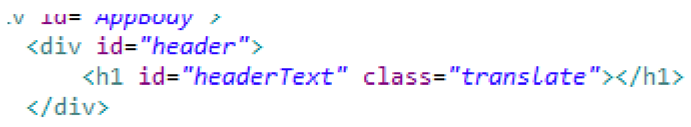
`messages.js` is the file that is designated for application strings and can be found in the `common/js` folder. If you use Dojo, jQuery, or Sencha Touch in your application, use the translation resource loading mechanisms and file formats from these JavaScript technologies instead of IBM Worklight. Use IBM Worklight application messages only when the JavaScript graphical toolkit used in your application does not provide these services.



```
Messages = {
    headerText: "Default header",
    actionsLabel: "Default action label",
    sampleText: "Default sample text",
    englishLanguage : "English",
    frenchLanguage : "French",
```

Application messages that are stored in `messages.js` can be referenced in two ways:

- As a JavaScript object property; for example, `Messages.header` or `Messages.sampleText`
- As the ID of an HTML element with `class="translate"`



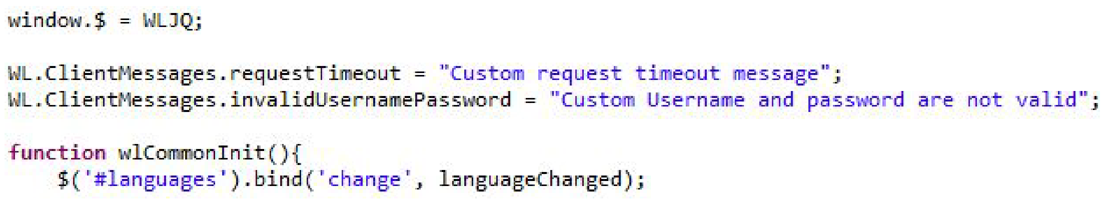
```
v id="AppBody" >
  <div id="header">
      <h1 id="headerText" class="translate"></h1>
  </div>
```

A string that is defined in `Messages.headerText` is automatically used here.

3.3 Enabling translation of system messages

You can also translate system messages that are displayed by the application; for example, “Internet connection is not available”, or “Invalid user name or password”.

- System messages are defined in the `WL.ClientMessages` object
- A complete list of system messages can be found in `wlclient/js/messages.js`, which is included in IBM Worklight generated projects
- Enable translation of a system message by overriding it in the application JavaScript



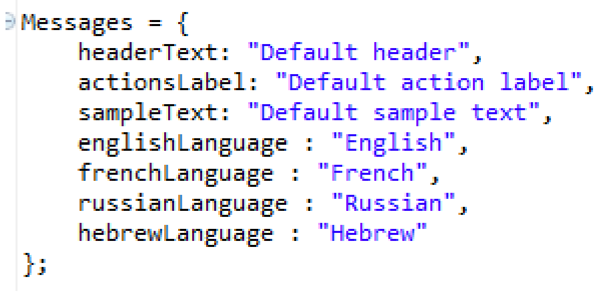
```
window.$ = WLJQ;

WL.ClientMessages.requestTimeout = "Custom request timeout message";
WL.ClientMessages.invalidUsernamePassword = "Custom Username and password are not valid";

function wlCommonInit(){
    $('#languages').bind('change', languageChanged);
```

Because some code is only run after the application has successfully initialized, override system messages at a global JavaScript level.

3.4 Implementing multi-language translation

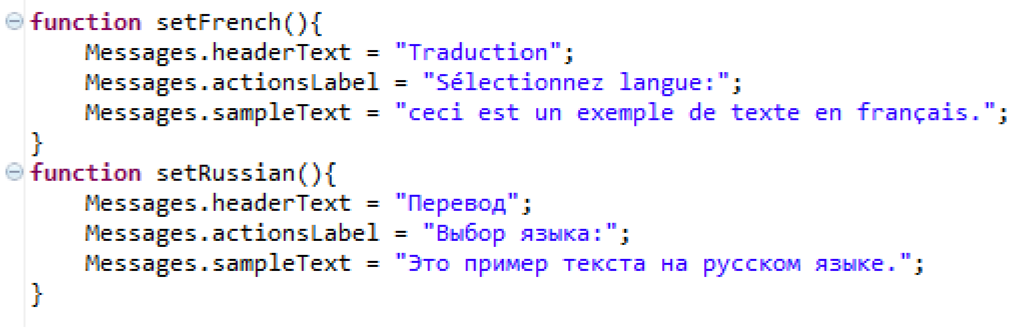
You can implement multi-language translation for your applications by using JavaScript.

1. Define default application strings in `messages.js` as shown in the following code snippet:

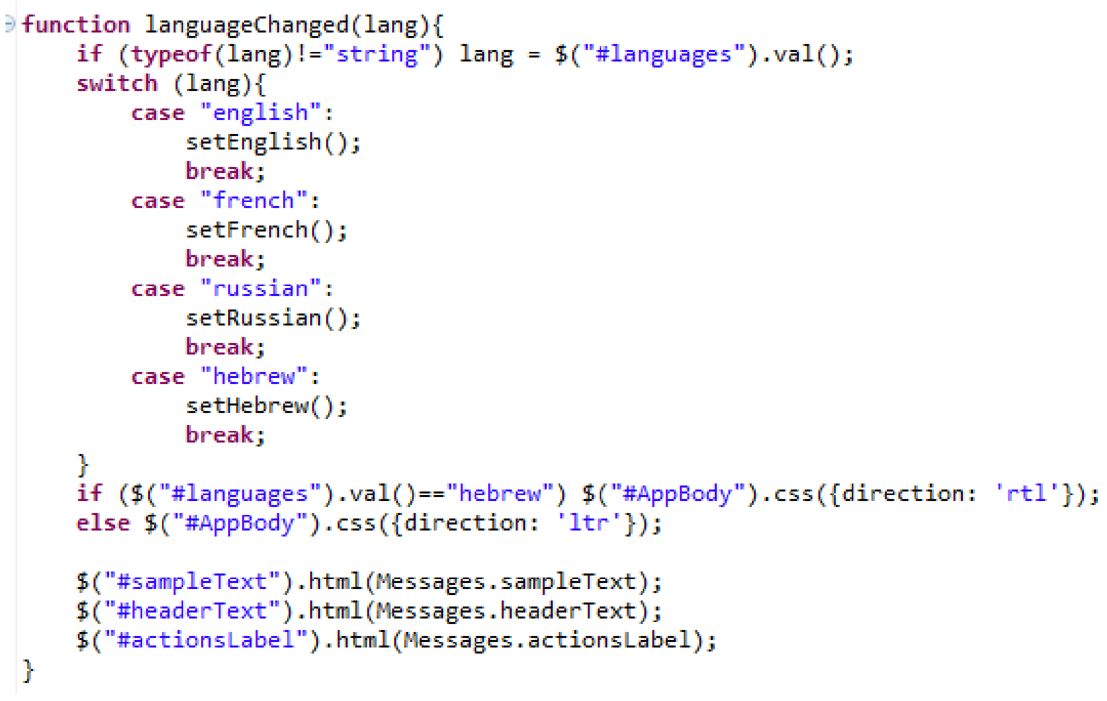


```
Messages = {
    headerText: "Default header",
    actionsLabel: "Default action label",
    sampleText: "Default sample text",
    englishLanguage : "English",
    frenchLanguage : "French",
    russianLanguage : "Russian",
    hebrewLanguage : "Hebrew"
};
```

2. Override some or all of the default application strings with JavaScript. The following two code snippets define JavaScript functions that are used to override the default strings that are defined in `messages.js`:



```
function setFrench(){
    Messages.headerText = "Traduction";
    Messages.actionsLabel = "Sélectionnez langue:";
    Messages.sampleText = "ceci est un exemple de texte en français.";
}
function setRussian(){
    Messages.headerText = "Перевод";
    Messages.actionsLabel = "Выбор языка:";
    Messages.sampleText = "Это пример текста на русском языке.";
}
```



```
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") $("#AppBody").css({direction: 'rtl'});
    else $("#AppBody").css({direction: 'ltr'});

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

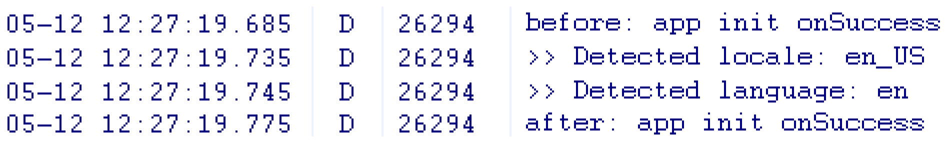
A language parameter is passed to the `languageChanged()` JavaScript function. The `languageChanged()` function calls the corresponding function to override the default English language string.

3.5 Detecting device-specific information

You can detect the locale and language of your device by using `WL.App.getDeviceLocale()` and `WL.App.getDeviceLanguage()`.

```
var locale = WL.App.getDeviceLocale();
var lang = WL.App.getDeviceLanguage();
WL.Logger.debug(">> Detected locale: " + locale);
WL.Logger.debug(">> Detected language: " + lang);
```

The following screenshot shows the print output:



```
05-12 12:27:19.685   D   26294   before: app init onSuccess
05-12 12:27:19.735   D   26294   >> Detected locale: en_US
05-12 12:27:19.745   D   26294   >> Detected language: en
05-12 12:27:19.775   D   26294   after: app init onSuccess
```

4 Advanced topics

4.1 Globalizing web services

In some situations, localized results are obtained by calling web services. The Cordova globalization method `getLocaleName` returns the user locale preference, which can be used in client-driven service calls.

Listing 14 shows how the user locale is used to collate a list of words. The locale of the returned word list can be checked to verify that the user locale was used or a substitute locale was used instead.

Listing 14: Locale-based service call

```
function getWords(){
  var services;
  require(
    ["dojox/rpc/Service"],
    function(Service){
      services = new Service({
        target: "http://9.191.81.210:8080/Services/JSON-RPC",
        transport: "POST",
        envelope: "JSON-RPC-1.0",
        contentType:
          "application/json",
          services: {
            "sorter.getWordList": {
              returns: {"type": "object"},
              parameters: [{"type": "string"}]
            }
          }
      });
    }
  );
  g11n.getLocaleName(
    function(locale){
      // invoke the JSON web service to get the list of sorted words
      var deferred = services.sorter.getWordList(locale.value);
      deferred.addCallback(
        function(result){
          var wordsView = dojo.byId("words");
          require(
            [
              "dojox/mobile/RoundRectList",
              "dojox/mobile/ListItem",
              "dojox/mobile/Heading"
            ],
            function(RoundRectList, ListItem, Heading){
              var wordsList = new RoundRectList({
                id: "words_list"}).placeAt(wordsView);
              items = result.words.list;
              for (var i = 0; i < items.length; ++i) {
                var word = new ListItem({label: items[i]});
                wordsList.addChild(word);
              }
              var wordsFooter = new Heading({
                label: result.localeName}).placeAt(wordsView);
            });
        }
      )
    },
    function(code){
      alert("Error: " + code);
    }
  );
};
```

4.2 Globalizing push notifications

Mobile applications frequently rely on server-side services to provide data to the mobile application. However, there are occasions when the application is not running or is not connected to the server. Push notifications are short messages that provide a mechanism for notifying mobile applications that a server has data that can be downloaded or information that can be viewed by the mobile application when the application is either not running or not running in the foreground.

Push notification currently works for iOS and Android only. iOS uses Apple Push Notification Service (APNS) to push notifications to mobile applications. Android uses Google Cloud Messaging (GCM) to push notifications to mobile applications. Translate push notification messages so that the correct language is displayed to the user. How the application runs, such as, in the foreground, in the background, or not running at all, determines your choice of architectural pattern.

- When the application is running in the foreground, it uses the language and cultural settings on the device to determine the appropriate language to display. To support this pattern, messages must be stored in the resource files of the mobile application, and not in the resource files of the server application, even though messages are generated on the server-side.
- When a notification is sent to a mobile application, send the notification resource key and not the actual text of the message.
- When a mobile application receives the notification, or message, use the key that was sent in the notification message to look up the text of the message from its resource file, as shown in the figure 4-1.

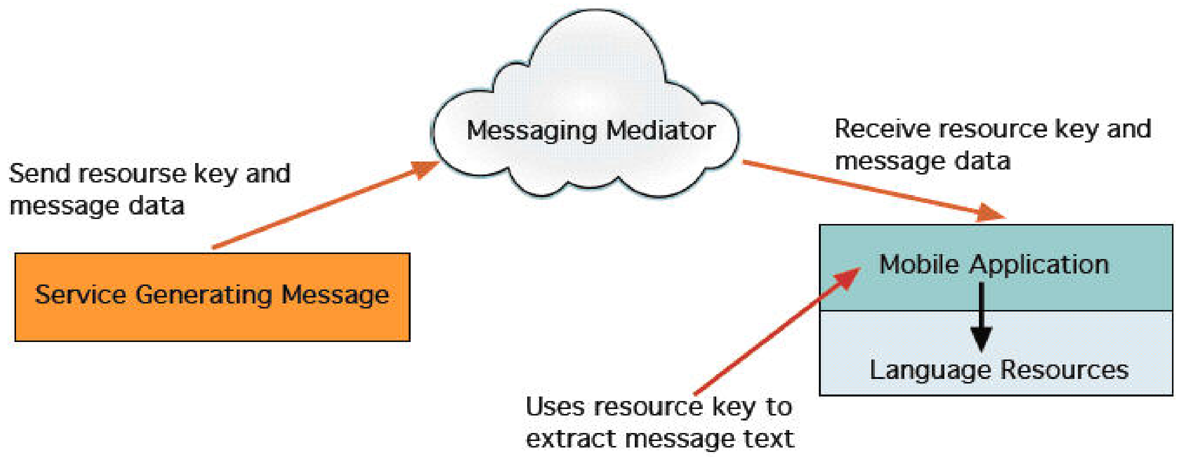

Messaging Mediator
Send resourse key and message data
Receive resource key and message data
Service Generating Message
Mobile Application
Language Resources
Uses resource key to extract message text

Figure 4-1: Using the resource key

Listing 15, **Listing 16**, and **Listing 17** show sample code that can be used when the mobile application is running in the foreground.

First, create an IBM Worklight adapter to send the notification. See **Module_41 - Push_Notifications** on http://www.ibm.com/mobile-docs for details on creating adapters.

Listing 15 shows how to send a notification with the created adapter. The target message, or resource key, to be translated on the client-side, is specified in the payload.

Listing 15: Send notification using created adapter

```
function sendNotificationOTA(userId, notificationText) {
  var userSubscription = WL.Server.getUserNotificationSubscription(
    'mysuranceAdapter.mysuranceEventSource', userId);
  WL.Server.notifyAllDevices(
    userSubscription,
    {
      badge : 1,
      sound : "",
      alert : notificationText,
      payload : { globalizeString : 'notificationText' }
    }
  );
}
```

Listing 16: Client-side subscription code

Listing 16 shows the code that is required on the client-side to subscribe to push notification.

```
WL.Client.Push.onReadyToSubscribe = function(){
  WL.Client.Push.registerEventSourceCallback(
    "mysurancePush",
    "mysuranceAdapter",
    "mysuranceEventSource",
    pushNotificationReceived);
};
```

After successful subscription, the callback method is implemented. The callback method is responsible for retrieving the data from the payload, retrieving application locale preferences, retrieving the message by using the resource key, and formatting the message.

Listing 17: Callback method

Listing 17 shows how to retrieve the locale information and load the corresponding translated message with Dojo by using the resource key that is stored in the payload object.

```
function pushNotificationReceived(props, payload){
  if (payload.globalizeString != "undefined"){
    require(
      ["dojox/mobile/i18n", "dojo/number"],
      function(mi18n, number){
        bundle = mi18n.load("resource", "messages");
        // get globalization text by dojo mobile i18n
        var notificationText = bundle[payload.globalizeString];
        // format number by device locale
        var num = number.format(1234567890, {
          places: 2, locale: WL.App.getDeviceLocale()});
        num = bundle["amount"] + num;
        //display globalization message
        alert(notificationText + "\n" +num);
      }
```

```
        );
    }
}
```

- If a notification provides data in addition to the message, then send the data in a locale-neutral format. When the application retrieves the message, the data can be formatted based on the user cultural preferences at the time the message is received.
- An application that is running in the background, or not running at all, can elect to use a previously registered user profile to access the desired language and cultural settings for push notifications. To support this pattern, the server sends the translated message and data in a format that is determined by the user cultural and language preferences that are stored in the profile, as shown in the figure 4-2. The push notifications are then processed differently by the mobile application. Processing is based on the native operating system that the application is running on.

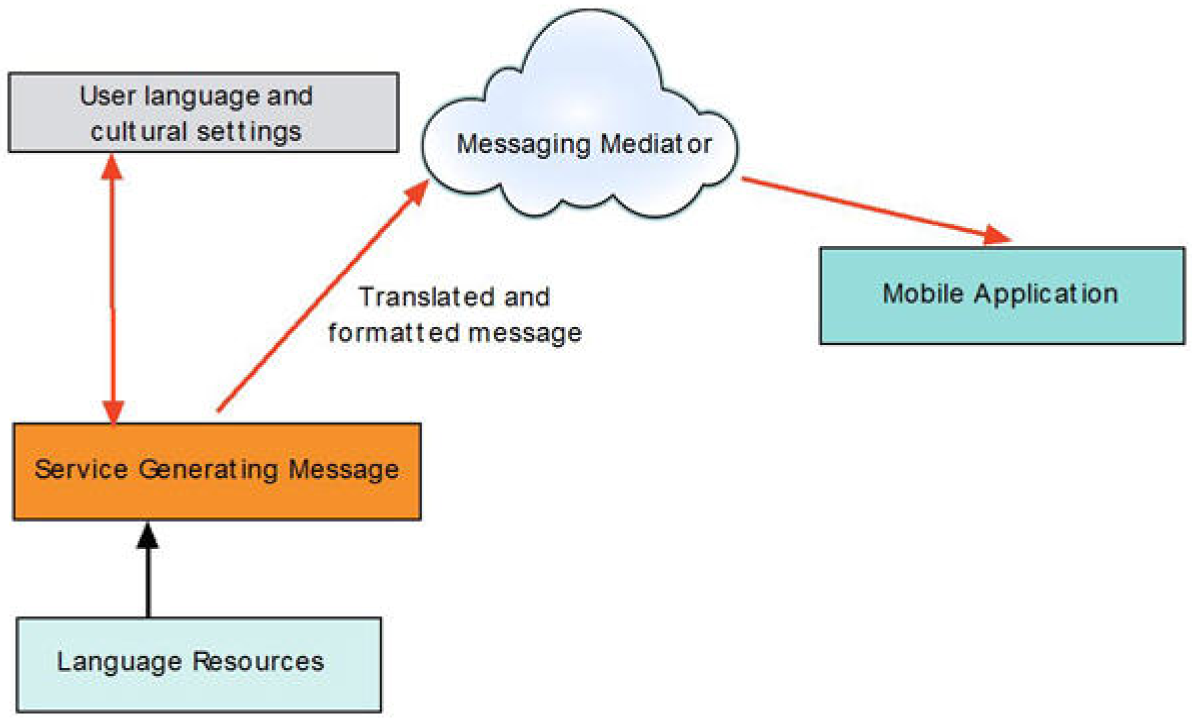

User language and cultural settings
Messaging Mediator
Mobile Application
Translated and formatted message
Service Generating Message
Language Resources

Figure 4-2: Sending push notifications according to the user's settings

- On Android, notification messages wake up Android applications, and the applications directly access the language and cultural preferences so that the correct translation and formatting can be applied.
- On iOS, notification messages do not wake up iOS applications, therefore the native operating system automatically selects the appropriate language to use for notifications. The iOS operating system automatically attempts to locate and load the correct language resource.
- In hybrid applications that are built using IBM Worklight, notifications are not directly processed by the application when the application is not running in the foreground. In this case, the

user language and cultural profile that was previously established is used.

Appendix A - Notices

Appendix B - Support and comments

For the entire IBM Worklight documentation set, training material and online forums where you can post questions, see the IBM website at:

http://www.ibm.com/mobile-docs

Support

Software Subscription and Support (also referred to as Software Maintenance) is included with licenses purchased through Passport Advantage and Passport Advantage Express. For additional information about the International Passport Advantage Agreement and the IBM International Passport Advantage Express Agreement, visit the Passport Advantage website at:

http://www.ibm.com/software/passportadvantage

If you have a Software Subscription and Support in effect, IBM provides you assistance for your routine, short duration installation and usage (how-to) questions, and code-related questions. For additional details, consult your IBM Software Support Handbook at:

http://www.ibm.com/support/handbook

Comments

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state.

Thank you for your support.

- Submit your comments in the IBM Worklight forums at:
- https://www.ibm.com/developerworks/mobile/mobileforum.html

If you would like a response from IBM, please provide the following information:

- Name
- Address
- Company or Organization
- Phone No.
- Email address