



# Orange County Flex Users Group

Irvine, CA

Oğuz Demirkapı

CTO  
NicheClick Media  
<http://NicheClick.com>

## I18N (INTERNATIONALIZATION) IN FLEX APPLICATIONS

# About Me

- CTO at NicheClick Media
- Coding since '85, CGI Programming since '94
- ColdFusion Developer since '97 (Flex since '07)
- Founder and Manager (prev) of CFUG for Turkey
- Interested in ColdFusion, Flex, AIR, Ajax, Frameworks, i18N, L10N, G11N
- Have big interest in Epistemology
- Living in Dana Point, CA
- Personal blog: <http://blog.demirkapi.net>

# Agenda

- What you may expect?
- i18N Theory & Basics
- Flex & i18N
- Resources
- Questions & Answers

# What You May Expect?

## Open Your Eyes, if;

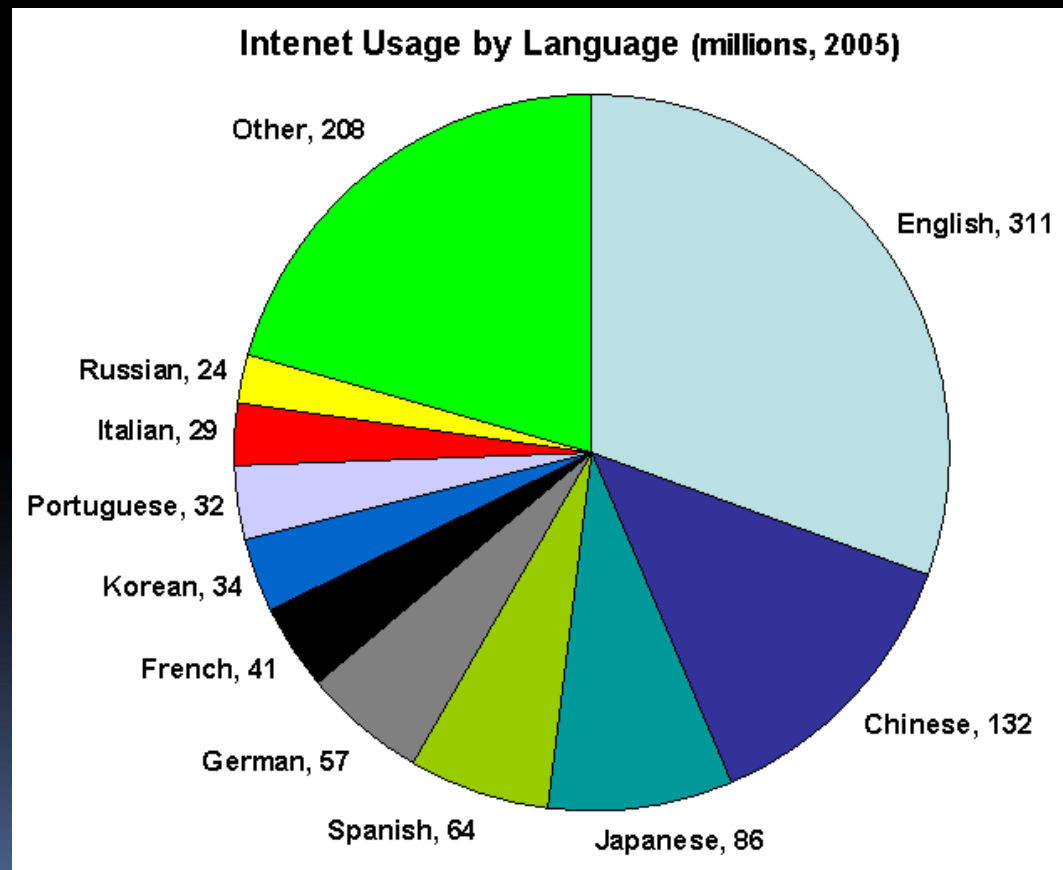
- You are a i18N newbie
- You heard that you should create i18N applications
- You need to get idea about i18N usage
- You want to get some tips

## May be not for you, if;

- You are already using i18N in Flex in production
- You think I know everything about i18N implementation on Flex 😊

# Do we need multilingualism?

- Yes!



# Why i18N (Internationalization)?

- The World Wide Web should be truly world-wide!
- Internationalization is important to ensure that users worldwide can equally benefit from Web technology
- Wide diversity world-wide:
  - Scripts (Latin, Cyrillic, Hebrew, Tamil, Katakana,...)
  - Languages (English, German, Turkish, Korean, Japanese,...)
  - Typographic conventions
  - Cultural conventions
  - Political circumstances
- Avoid fragmentation of specifications due to localization
- Make sure internationalization is done at the right place

# What is ...?

- What is i18N (Internationalization)?
  - Application functions in at least two locales
- What is L10N (Localization)?
  - Process of applying a locale or language "skin" to an i18N application
- What is G11N (Globalization)?
  - i18N & L10N

# Globalization & Localization

## Globalization

- Single character set
- Single executable
- Single install
- Single server serves all clients in all languages

## Localization

- Based on globalized software
- Adds specific translations and adaptations for particular languages and markets

Globalized software can be localized without any code changes

# Character encoding

Character encoding specifies mappings from a character set to the integer numbers that represent the characters on a computer.

- EUC-JP (Japanese)
- EUC-KR (Korean)
- ISO-8859-1 (Western European and English)
- SHIFT\_JIS (Japanese)
- UTF-8 (All Languages)

# What is Locale?

- The combination of a language and a country code

en\_US: US English (color, \$)

en\_GB: British English (colour, £)

de\_CH: German in Switzerland

tr\_TR: Turkish in Turkey

# Selecting Locale

- Manual
  - Default locale can be loaded and other options would be available by selection via buttons/selects etc.
- Locale Detection
  - Capabilities.language property in Flash Player and AIR
  - Parsing 'Accept-Language' header on HTTP request
    - Not usable with URLLoader
  - Parsing browser and OS language settings via JavaScript
  - Location detection depending on IP

# Problems

- Charset Conversions
- Formatting & Parsing
  - Date & time
  - Messages
  - Numbers & currencies
- Translated Names
  - Languages, Regions (Countries), Scripts, Time zones, Currencies
- Calendar, Time Zone, Date/Time conversions
- ▶ Collation
  - Searching, Sorting, Matching
- ▶ Segmentation
  - word, line, ...
- ▶ Transforms
  - Normalization
  - Casing
  - Transliterations
- ▶ Unicode Regular Expressions
- ▶ Complex-Text Display / Input
- ...

# What is Unicode?

- Unicode (unicode.org) is an character set for all the characters and symbols of the world.
- Unicode provides a unique number for every character.
- Except Klingon! 😊

ما هي الشفرة الموحدة "يونكود" ؟

Unicode nedir?

Τι είναι το Unicode

Cos'è Unicode?

유니코드에 대해?

Что такое Unicode?

# Unicode (cont.)

- Why do we need to use Unicode?
  - Avoids data corruption
  - Single encoding for text in all languages
  - Makes software globalization possible
    - Vastly reduces development cost
    - Vastly reduces maintenance, update and support cost
- Switching to Unicode has no disadvantages for single language users, to the contrary it usually offers advantages even for single language users. And it offers great advantages for multilingual users.
- Encoding: Use Unicode wherever possible for content, databases, etc. Always declare the encoding of content.

# Unicode (cont.)

- Non-Globalized Component
  - Does not use Unicode
  - Hard-coded date/time formatting & parsing
  - Hard-coded number & currency formatting & parsing
  - Hard-coded collation (sorting/searching/matching)
  - Other hard-coded operations
  - Hard-coded literals

# Unicode & Files

- UTF-8 is the recommended encoding for files.
- Use a Unicode capable editor (IDE)
- Flex Builder
  - Default encoding is UTF-8
- Eclipse
  - Default encoding is Cp1252
  - Change it into UTF-8
    - Window-> Preferences -> General -> Workspace
      - Text file encoding

# Unicode & Database

- Use a robust database with right settings
  - MS SQL Server
  - MySQL Server
  - PostgreSQL Server
  - Oracle
- Beware of Unicode Support
  - MySQL 4.1 and up
    - default-character-set=utf8
  - MS SQL Server
    - SQL Server nvarchar, ntext etc.

# Flex & i18N

- Flex 3 has embedded i18N features
- It is possible to create i18N applications in Flex 2
  - Using ResourceBundle class
  - Different approaches such as XML usage etc.

# Localization in Flex

- Resource Bundles
  - Idea is based on Java resource bundles technique
  - Every locale has a localized version of properties files
  - Creating localized files may need resource bundle managers
    - Attesoro ([attesoro.org](http://attesoro.org))
    - Eclipse Plug-in ([sourceforge.net/projects/eclipse-rbe/](http://sourceforge.net/projects/eclipse-rbe/))
    - RBMan (Flex Application) ([rbman.riaforge.org](http://rbman.riaforge.org))
  - Binary assets such as audio files, video files, SWF files, and images can be referenced
    - `flag=Embed("images/flag_us.png")`
- Every locale needs to create localized properties files

# Localization in Flex - 2

- Properties files can be compiled as resource bundles

OR

- Properties files can be loaded at run time by resource modules

# Localization in Flex - 3

- Flex lets you change locales on the fly
  - If you compile more than one resource bundle into an application, you can toggle the resource bundle based on the locale
- If you compile resource modules
  - You can load and unload the SWF files at run time based on the locale
  - You can create new resource bundles programmatically
- If your application supports many locales, you will likely want to load the appropriate resources at run time rather than compile all supported resources into the application at compile time.

# Accessing The Values in Resource Bundles

- Bind values from the resource to an expression
  - @Resource directive
  - Available only in MXML
  - Cannot change locales at run time
  - Only returns Strings
- Methods of the ResourceManager class
  - Can be used also in ActionScript
  - Can return data types other than Strings, such as ints, Booleans, and Numbers (getNumber(), getBoolean())
  - Can be used to switch locales at run time
  - Not supported in Flex Builder or ANT FlexTasks and needs to use command line

# Accessing The Values in Resource Bundles

- Compiling as Runtime Shared Libraries
  - Saves re-compiling your bundles every time while compiling the main application
- Compiling into Component Modules
  - Separation of dependencies with the main application
- Manually loading into the resourceManager at runtime
  - Remoting can be used to load the required resources from the server
  - Have to have a connection to the server retrieve the bundles
  - Modifying the resource bundles on the server without having to re-compile / re-deploy the application is possible

# Using the @Resource Directive

- Sample

```
<mx:FormItem label="@Resource(key='username',  
    bundle='myRB') ">  
    <mx:TextInput />  
</mx:FormItem>
```

- Compile Parameters

```
-locale=en_US -allow-source-path-overlap=true -source-path=locale/{locale}
```

- Using the ResourceManager

```
<mx:Metadata>  
    [ ResourceBundle("myRB") ]  
    [ ResourceBundle("myOtherRB") ]  
</mx:Metadata>
```

# Using the Resource Classes

- On an ActionScript

```
<mx:Script><![CDATA[  
import mx.resources.ResourceBundle;  
import mx.controls.Alert;  
private function registrationComplete():void {  
    Alert.show(resourceManager.getString('myRB',  
    'thanks'));  
} ]]>  
</mx:Script>
```

# Using a Class

- Can be used for programmatic skin
- The following example embeds the MyCheckBoxIcon\_en\_US class in the properties file
  - CHECKBOXSKIN=ClassReference("MyCheckBoxIcon\_en\_US")
- Binding the value of the getClass() method for programmatic skins

```
<mx:CheckBox selected="true"  
  selectedUpIcon="{resourceManager.getClass('bundle1','CHECKBOXSKIN')}}"  
  selectedDownIcon="{resourceManager.getClass('bundle1','CHECKBOXSKIN')}}"  
  selectedOverIcon="{resourceManager.getClass('bundle1','CHECKBOXSKIN')}}"  
/>
```

# Changing Locale in Runtime

- To be able to change the locale at run time without using resource modules, you compile all the available locales into the application at compile time by including them as part of the locale option
  - `-locale=en_US,es_ES`
- The Flex application uses the list of locales in the `localeChain` property to determine precedence when getting values from resource bundles. If a value does not exist in the first locale in the list, the Flex application looks for that value in the next locale in the list, and so on.
  - `localeComboBox.selectedIndex = locales.indexOf(resourceManager.localeChain[0]);`

# Formatting Date & Times

- DateFormatter and CurrencyFormatter can use resources

```
THEMECOLOR=0x0000FF  
DATE_FORMAT=MM/DD/YY  
TIME_FORMAT=L:NN A  
CURRENCY_PRECISION=2  
CURRENCY_SYMBOL=$  
THOUSANDS_SEPARATOR=,  
DECIMAL_SEPARATOR=.
```

```
<mx:DateFormatter id="dateFormatter"  
    formatString="{resourceManager.getString('FormattingValues',  
    'DATE_FORMAT')}}" />
```

# Styles in Localized Resources

- Binding can be used  
`themeColor="{resourceManager.getUint('MyStyles', 'color')}"`
- If the value is a color like `0xFF0000`, use the `ResourceManager`'s `getUint()` method
- If the value is a class, like a programmatic skin, then use the `getClass()` method

# Fonts in Localized Resources

- Resource Files

```
# /locale/en_US/FontProps.properties
```

```
TEXTSTYLE=myENFont
```

```
# /locale/ja_JP/FontProps.properties
```

```
TEXTSTYLE=myJAFont
```

- Application

```
<mx:Style>
```

```
@font-face { src: url("ENFont.ttf"); fontFamily: EmbeddedENFont; } @font-face { src: url("JAFont.ttf"); fontFamily: EmbeddedJAFont; } .myENFont{ fontFamily: EmbeddedENFont; }
```

```
.myJAFont{ fontFamily: EmbeddedJAFont; }
```

```
</mx:Style>
```

- In MXML

```
<mx:Text styleName="{resourceManager.getString('FontProps', 'TEXTSTYLE')}" />
```

# AIR & i18N

- Generating via Flex
  - Flex i18N can be used
- Generating via HTML/JavaScript (Ajax)
  - General techniques which are available in required platform can be used
  - Be careful on data transaction such as on JSON etc.
- SQLite
  - SQLite database can be used to customize assets and literals

# Resources

- Unicode

<http://unicode.org>

- Flex 3 Documentations

<http://www.adobe.com/support/documentation/en/flex/>

[http://livedocs.adobe.com/flex/3/html/l10n\\_1.html](http://livedocs.adobe.com/flex/3/html/l10n_1.html)

- Flex.org

- RBMan

<http://rbman.riaforge.org>

- My blog

<http://blog.demirkapi.net>

# Questions & Answers



Oğuz Demirkapı

<http://blog.demirkapi.net>

[OCFlex@demirkapi.net](mailto:OCFlex@demirkapi.net)