Android & iPhone Localization

Amir Eibagi

Topics

- **Android Localization:**
  - Overview
  - Language & Strings
  - Country/region language variations
  - Images & Media
  - Currency, date & Time
- **iPhone Localization**
  - Language & Strings
  - Country/region language variations
  - Images & Media
  - Currency, date & Time

Big Idea

- **Good news**
  - – Android's use of resource files (res/layout/main.xml, res/values/strings.xml, etc.) simplifies GUI development
- **Bad news**
  - – Descriptions in English won't work in German
  - – What fits in portrait orientation won't fit in landscape
  - – Images for high-density screens are too large for low-density ones
- **Solution**
  - – Make multiple layout and resource files
    - For different language, orientations, etc.
  - – Have Android automatically switch among or combine them
- **Notes**
  - – Localization sometimes called L10N (L, 10 letters, N)
  - – Also sometimes called Internationalization (I18N)

Process

- **Make qualified versions of resource files**
  - – Find the settings that affect your application
    - Language, orientation, touchscreen type, dock mode, etc.
  - – Find qualifier names that correspond to each setting
    - Language: en, en-rUS, es, es-rMX, etc.
    - Screen orientation: port, land
    - Display density: xhdpi, hdpi, mdpi, ldpi
    - Dock mode: car, desk
    - Etc.
  - – Append qualifier names to folder names
    - res/values/strings.xml, res/values-es/strings.xml, res/values-es-rMX/main.xml
    - res/layout/main.xml, res/layout-land/main.xml
- **Load the resource normally**
  - – R.string.title, R.layout.main, etc.
  - – Android will switch among layout files automatically
  - – Android will combine values files automatically

Language and Strings

Steps

- **Make multiple folders with language codes**
  - – res/values, res/values-es, res/values-ja, etc.
    - Language codes are specified by ISO 639-1
  - – http://en.wikipedia.org/wiki/ISO_639-1
- **Define *all* strings in default folder**
  - – In res/values, define *all* names
    - Use the most common language
  - – E.g., res/values/strings.xml (or other name in res/values)

    <string name="company_name">Apple</string>

    <string name="welcome_message">Welcome!</string>
- **Use similar approach for colors, images, etc.**
  - – Use res/values/ for *all* colors, dimensions, arrays, etc.
  - – Use res/drawable for *all* image files
  - – Use res/raw for *all* audio and video files

Steps (continued)

- **Put language-specific strings in languagespecific folders**
  - – In res/values-es/strings.xml (or res/values-ja, etc), redefine
    - *only* the names that change based on language
  - – E.g., in res/values-es/strings.xm
    - <string name="welcome_message">¡Bienvenidos!</string>
      - – No entry for company_name, since the company name does not change (in Spanish, it is still Apple, not Manzana)
  - – E.g., in res/values-ja/strings.xml
    - <string name="welcome_message">ようこそ！</string>
      - – No entry for company_name, since the company name does not change (in Japanese, it is still Apple, not アップル)
- **Use similar approach for other resources**
  - – res/values-es/colors.xml, res/drawable-es/flag.png, etc.
    - *Only* redefine the ones that change based on language

Steps (Continued)

- **In XML, refer to base string name**
  - – someAttribute="@string/company_name"
  - – someAttribute="@ string/welcome_message"
    - No reference to folder or language.
    - Android will provide the proper version automatically. It *first* loads values from res/values/strings.xml, *then* loads values from res/values-es/strings.xml. Any names in second file that are common to first file are replaced.
- **In Java, refer to base string name**
  - – getString(R.string.company_name)
  - – getString(R.string.welcome_message)
    - No reference to folder or language. Same process as above.
- **Use similar approach for other resources**
  - – XML: @drawable/flag, @color/default_foreground, etc.
  - – Java: R.drawable.flag, R.color.default_foreground, etc.

How User Changes Device Language

- **On phone (or other physical Android device)**
  - – Go to home screen, press Menu button, select Settings
    - (Most people also have the Settings app on desktop)
  - – Choose Language and Keyboard
  - – Choose Select locale at the top
  - Most phones will have a very limited number of choices, based on what device manufacturer supports
  - – Android cannot (easily) use localization within apps unless entire OS supports that language.

Example: The Android Resort

- **Idea**
  - – Make an app that advertises a luxury resort where visitors can sit inside all day and play with their smart phones
- **Approach**
  - – res/values/strings.xml defines resort_name, welcome, our, pool, reserve, confirmed
  - – res/values-es/strings.xml defines welcome, our, pool, reserve, confirmed
    - – Does not redefine resort_name
  - – Also uses dimensions, colors, images, and layout files
    - • But these do not change based on language,

Strings File: English/Other (res/values/strings.xml)

- <?xml version="1.0" encoding="utf-8"?>
- <resources>
- <string name="resort_name">AndroidResort.com</string>
- <string name="welcome">Welcome to </string>
- <string name="our">Our </string>
- <string name="pool">swimming pool</string>
- <string name="reserve">Reserve Now!</string>
- <string name="confirmed">Registration Confirmed</string>
- </resources>

Strings File: Spanish (res/values-es/strings.xml)

- <?xml version="1.0" encoding="utf-8"?>
- <resources>
- <string name="welcome">Bienvenido a </string>
- <string name="our">Nuestra </string>
- <string name="pool">piscina</string>
- <string name="reserve">¡Reserva Ahora!</string>
- <string name="confirmed">Registro Confirmado</string>
- </resources>

Strings File: Mexican Spanish (res/values-es-rMX/strings.xml)

- <?xml version="1.0" encoding="utf-8"?>
- <resources>
  - <string name="pool">alberca</string>
- </resources>

Layout File res/layout/main.xml (No Language-Based Versions)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://..." android:orientation="vertical" ...>

    <TextView android:text="@string/welcome" ... />
    <TextView android:text="@string/resort_name" ... />

    <ImageView android:src="@drawable/android_resort_pool"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:adjustViewBounds="true"
    android:scaleType="fitXY"/>
<LinearLayout android:gravity="center_horizontal"
android:layout_height="wrap_content"
android:layout_width="match_parent">

    <TextView android:text="@string/our" ... />
    <TextView android:text="@string/pool" .../>
</LinearLayout>
    <Button android:text="@string/reserve" .../>

</LinearLayout>
```

Manifest File (No Language-Based Versions)

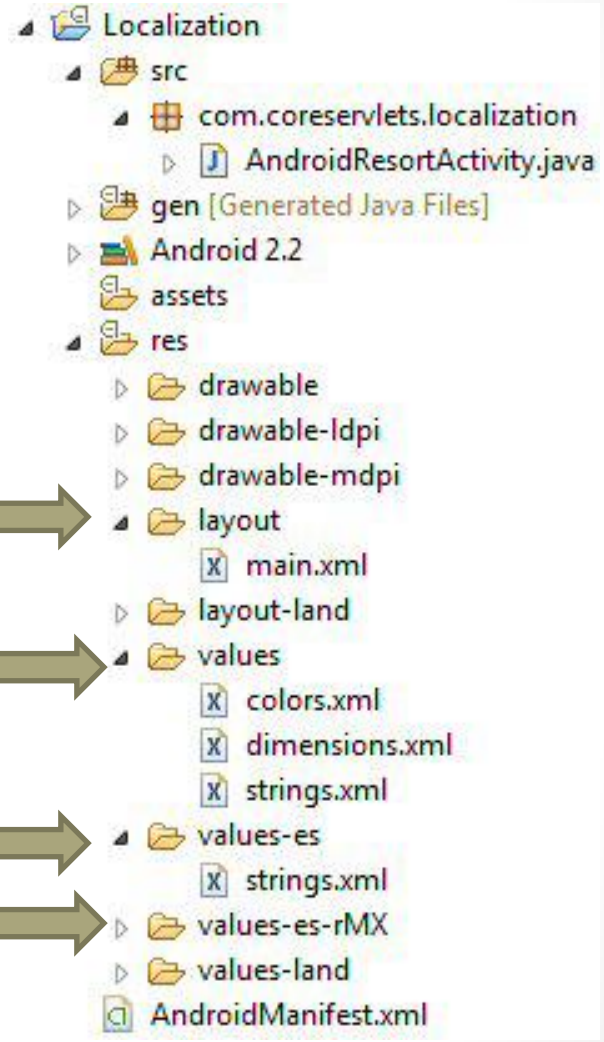
- <?xml version="1.0" encoding="utf-8"?>
- <manifest xmlns:android="http://schemas.android.com/apk/res/android"
- package="com.coreservlets.localization"
- android:versionCode="1"
- android:versionName="1.0">
- <uses-sdk android:minSdkVersion="8" />
- <application android:icon="@drawable/icon"
  - android:label="@string/resort_name">
- <activity android:name=".AndroidResortActivity"
  - android:label="@string/resort_name">
- <intent-filter>
- <action android:name="android.intent.action.MAIN" />
- <category android:name="android.intent.category.LAUNCHER" />
- </intent-filter>
- </activity>
- </application>
- </manifest>

Java

```
public class AndroidResortActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
public void confirmRegistration(View clickedButton) {
String message = getString(R.string.confirmed);
showToast(message);
}
private void showToast(String text) {
Toast.makeText(this, text, Toast.LENGTH_LONG).show();
}
}
```

Layout

- Default Layout
- Default (English) Strings
- Spanish Strings
- Spanish strings
  - (Depends on region)



Results: English VS. Spanish

Best Practices

- **Provide unlocalized defaults for all values**
  - So if Locale is unexpected, it displays in default language
- **Use graphical layout editor for testing**
- **Avoid changing layouts based on language**
  Might be necessary in some cases (e.g., US English asks for given name first and family name second, whereas Indian English asks for them in opposite order). However,
  - makes for hard-to-maintain code.
    - Consider putting the logic in Java code instead

Changing the Language Programmatically: Code

- **Steps**
  - Locale locale = new Locale("es"); *// Language code*
  - Locale.setDefault(locale);
  - Configuration config = new Configuration();
  - config.locale = locale;
  - context.getResources().updateConfiguration(config, null);
    - – context above is reference to the main Activity

Date& Currency

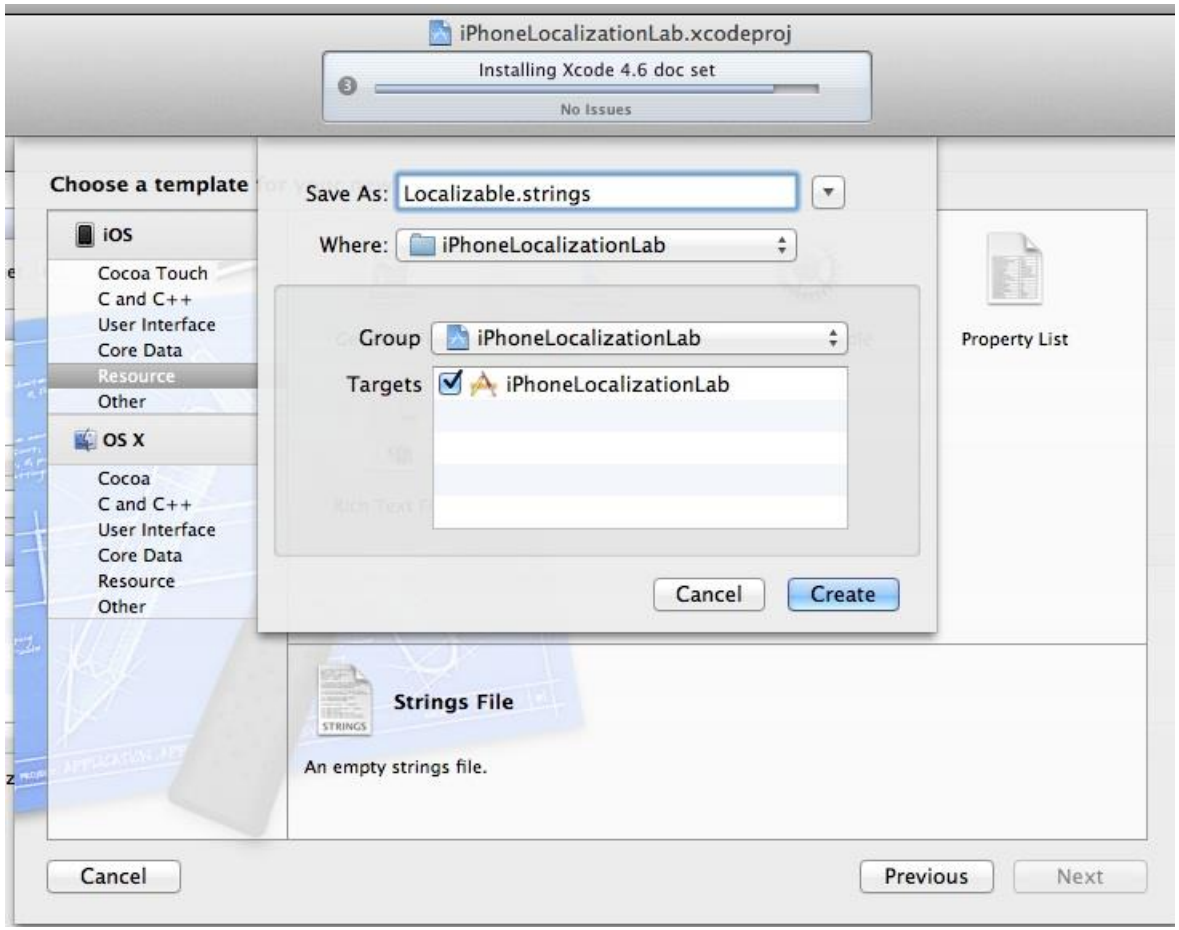
- Could be done using java's DateFormat & Currency Classes
- Time & Date:
  - String currentDateTimeString = DateFormat.*getDateTimeInstance().format(**new Date());*
    - *Date Format depends on the region*

- Currency Symbol:
  - Locale current = getResources().getConfiguration().locale;
  - Currency currency= Currency.*getInstance(current);*
  - String symbol = currency.getSymbol();

iPhone Localization

Steps (String & text):

- Create a new string file “localizable.strings”



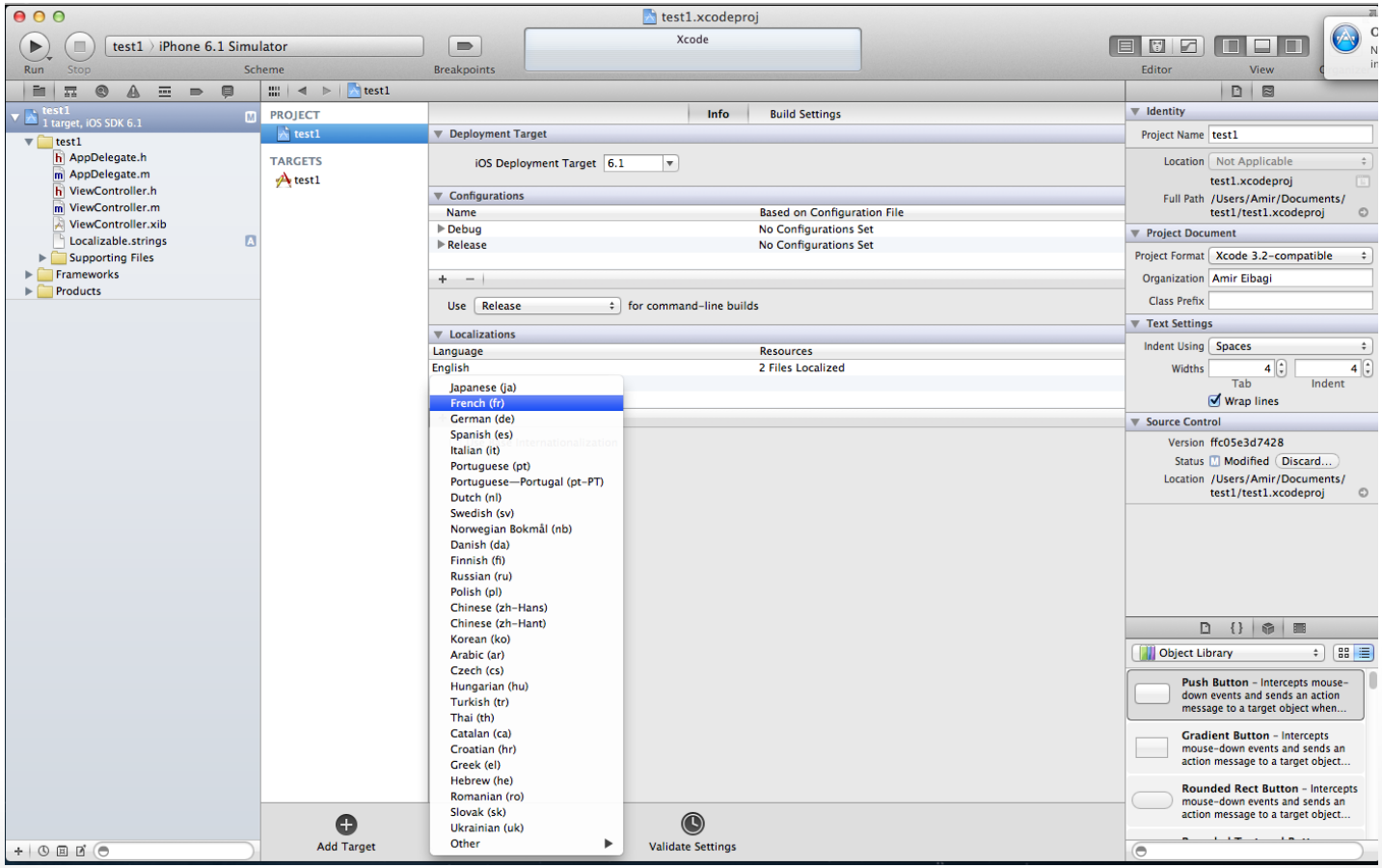
Steps (String & text):

- Strings are stores in the “key-value” format:

- "Welcome" = "Welcome to the United States!";
- "ShowInfo" = "Show Info";
- "dateTime" = "Date and Time:";
- "Currency" = "Currency is Dollars: $";

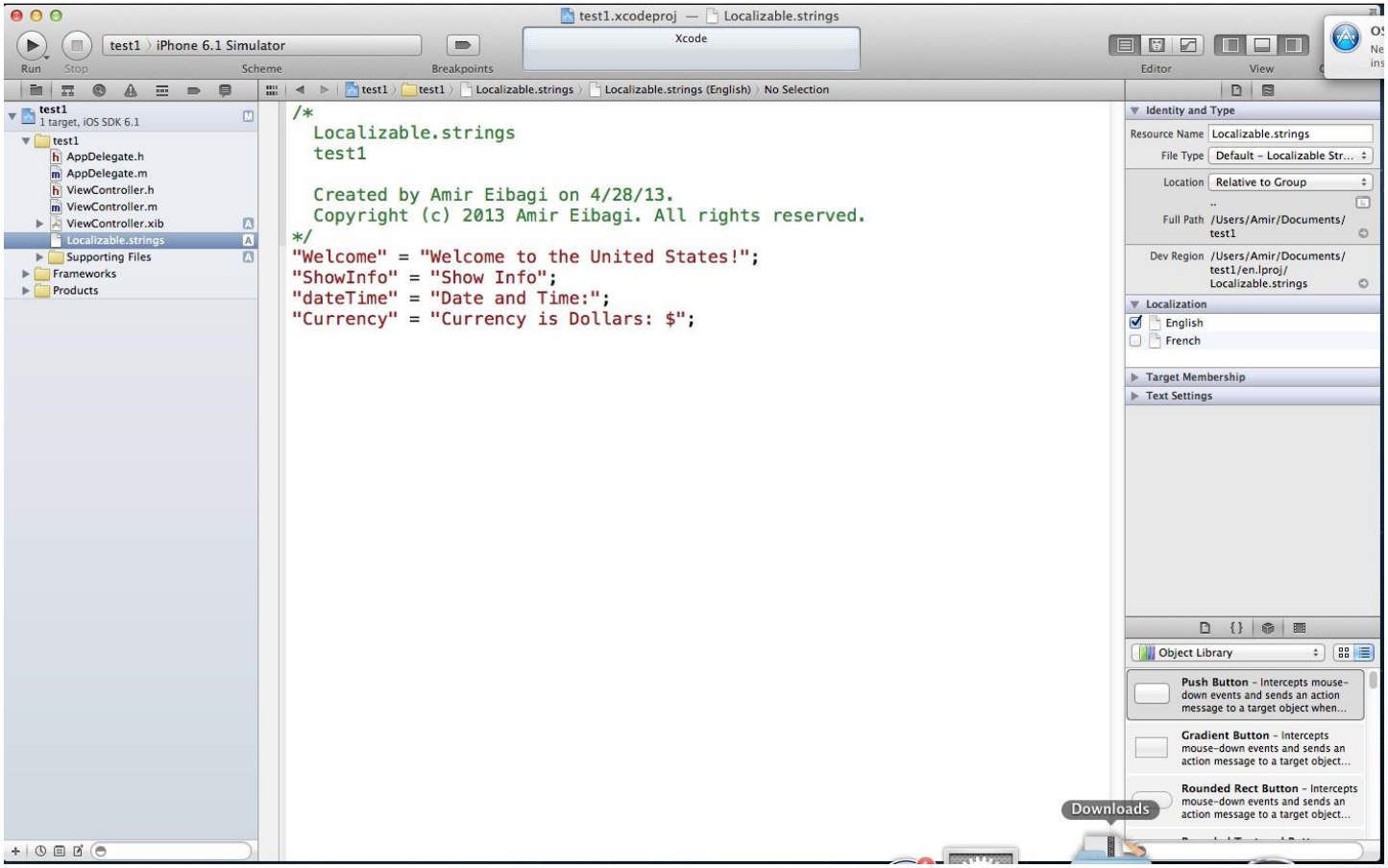
Steps(continued)

- Now click on the project folder on the project navigator and select your project's name under "related files" and click on the "info" tab.
- Click "+" sign under localization and add a language:



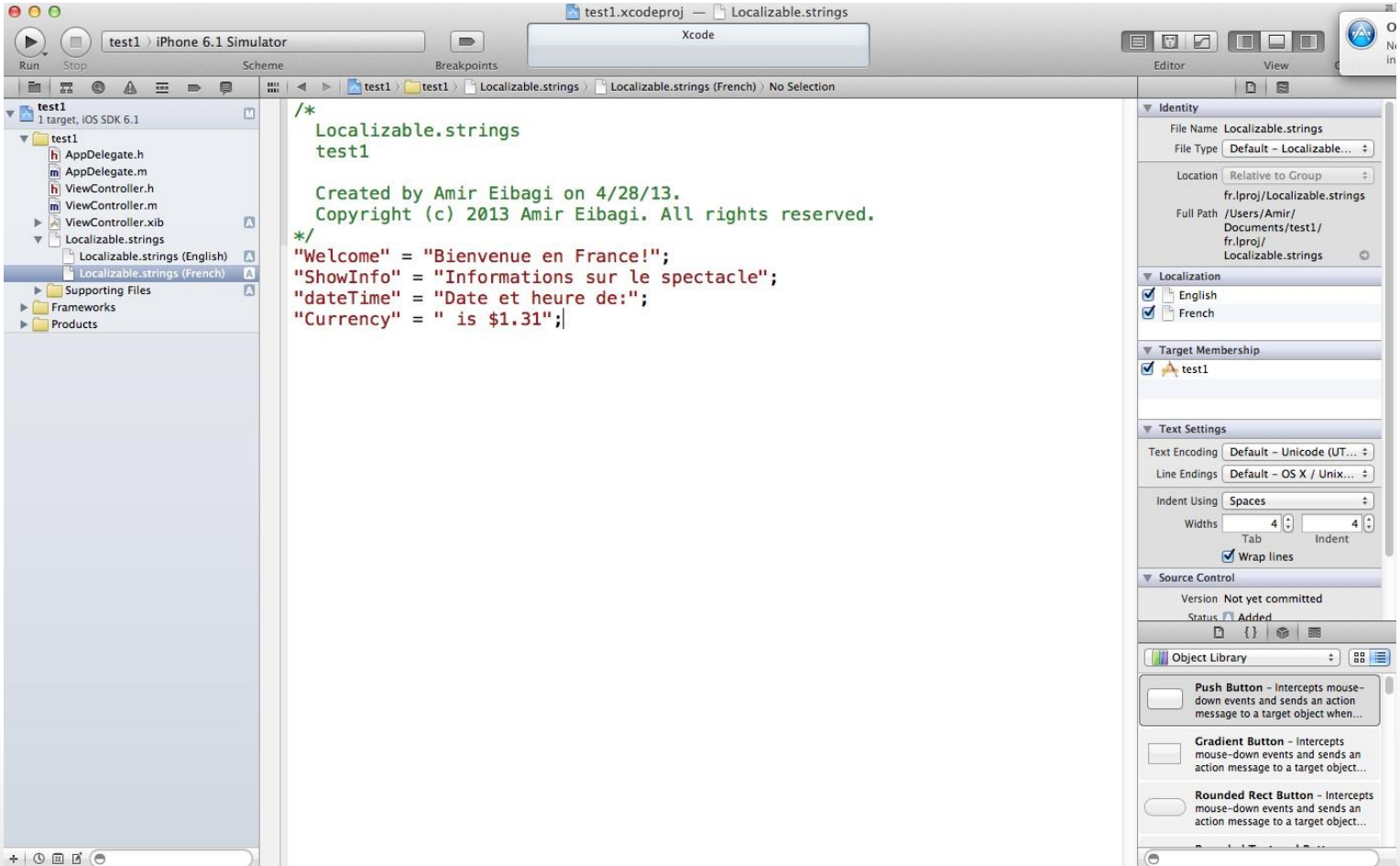
Steps(continued)

- Select Localizable.strings and click on “Localize..” under localization in file inspector



Steps(continued)

- Add French and look under project inspector for localizable.strings (French)
- Add French text to this file.



Access String from Code

- Find the keys from localizable.strings to get its value:

- welcome_label.text = [NSString stringWithFormat:NSLocalizedString(@"Welcome", nil)];

Images

- Repeated the steps as strings for images.
- Place an image inside your project directory, click on localize and add select languages that you have previously added to your project.
- Make sure all the localizable images have the same name. (Same rule also applies for the Sting keys under localizable.strings)

- **UIImageView *im = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"flag.png"]];**

Date & Time

- Use NSString with NSDateFormatter

- Setup the date format using the current locale:
- **NSString *dateComponents = @"yMMd";**
- **NSString *dateFormat = [NSDateFormatter dateFormatFromTemplate:dateComponents options:0 locale:[NSLocale currentLocale]];**

- Setup the time format as follows:
- **NSString *newStr = [[NSString alloc] initWithFormat:@"%@ H:mm:ss", dateFormat];**
- **NSDateFormatter *formatter = [[NSDateFormatter alloc] init];**
- **[formatter setDateFormat:newStr];**
- **NSString *formattedDateString = [formatter stringFromDate:[NSDate date]];**
- Device Region must be changed in order to see the correct Date format

Currency Symbol

- Use NSNumberFormatter
- Automatically picks the currency symbol that matches the device's region

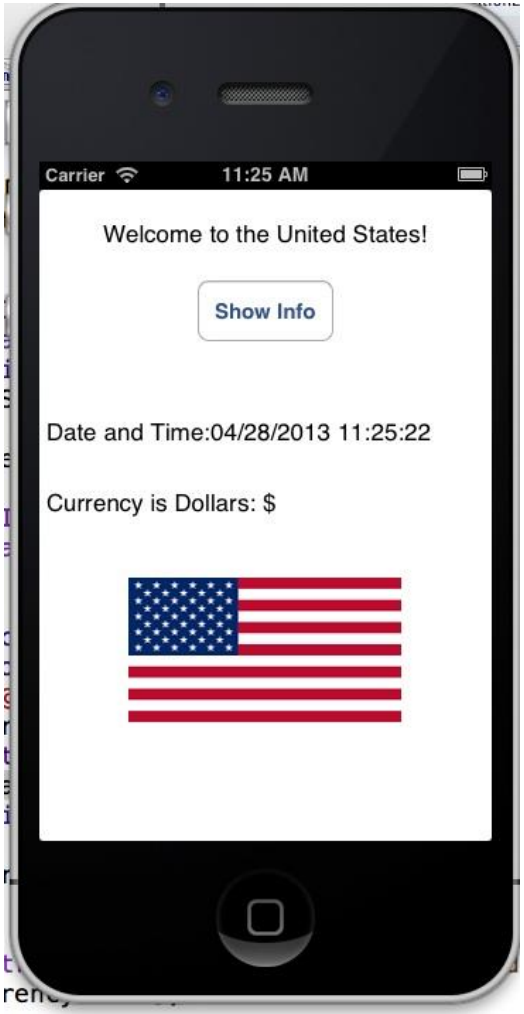
- **NSDecimalNumber *someAmount = [NSDecimalNumber decimalNumberWithString:@"1.00"];**

- **NSNumberFormatter *currencyFormatter = [[NSNumberFormatter alloc] init];**
- **[currencyFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];**
- **[currencyFormatter stringFromNumber:someAmount]**

- Device Region must be changed in order to see the correct currency symbol

Result

- Default language is set to English (United States)



Change Device language & Region

- Change both language and region to see the correct currency symbol and date format:

Result (French)