Tema 9 - Internacionalización (I18N) y Localización (L10N) (DCA)

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE

Índice

Logo DLSI

Tema 9 - Internacionalización (I18N) y Localización (L10N)
Curso 2018-2019

1. ¿A qué llamamos "*locale* " ?

- A lo que hace que funcione la *internacionalización* (I18N).
- Consta de una serie de parámetros culturales escritos en un archivo, en una variante de un idioma hablado en un territorio.
- Estos parámetros incluyen **el juego de caracteres empleado**, **el código del lenguaje**, la representación de **fecha y hora**, **números**, **moneda**, **direcciones**, **teléfono** y **medidas**.

```
idioma[_territorio][.codeset][@modifier]
```

- Ejemplos: **en_GB**, **en_NZ**, **es_MX**, **ca**, **ca@valencia**.
- La parte del idioma se codifica en `iso-639`, la del territorio en `iso-3166` y el archivo con los datos en `iso-15924`.
- Cada parámetro puede tener asociado un locale diferente, podemos tener los mensajes en un idioma, la representación de la moneda de otro, etc...
- Los ajustes del *locale* se suelen hacer para cada usuario.
- Por defecto se usa el *locale* de "`C`" (locale POSIX).

2. I18N vs. L10N

- La **internacionalización** de un proyecto consiste en prepararlo de forma que sea capaz de trabajar y presentarse en una multitud de idiomas.
- La **localización** toma un programa previamente *internacionalizado* y le proporciona la suficiente información para que se adapte al idioma y configuración del usuario actual.

3. Información del "locale" actual

- Comprobamos los locales disponibles.

```
1        locale -a
2
3        # El sistema nos respondera con algo similar a esto...
4        C
5        ca_ES.utf8
6        C.UTF-8
7        es_ES.utf8
8        POSIX
```

- En Sistemas Operativos de la familia GNU/Linux tenemos las locales disponibles en el archivo "`/etc/locale.gen`". Es un fichero de texto que se edita como administrador.
- Luego debemos ejecutar: "`sudo locale-gen`".
- Una prueba de cambio temporal al locale *catalán*: "`LANG=ca_ES.utf8 cal`"

4. GNU Gettext

- Es un marco de trabajo y un conjunto de herramientas que permiten internacionalizar un proyecto.
- Empleado por la mayoría de proyectos de software libre.
- El ajuste del idioma (y del resto de locales) se deduce de la elección del idioma hecha por el usuario para su sesión de trabajo.

5. *I18n* de un proyecto (I)

- Creamos un directorio "`po`" en la raíz del proyecto.
- Extraemos las cadenas a traducir de los archivos de código fuente: **xgettext**:

```
xgettext -d intlapp -o po/intlapp.pot -s ./src/app.vala
```

- Esto genera un archivo llamado: `intlapp.po`. Lo copiamos a un fichero llamado según el código del idioma al que lo queramos traducir, por ejemplo *catalán*: **ca.po**
- También podemos realizar este último paso de este modo: **msginit**:

```
msginit -l es -o po/ca.po -i po/intlapp.pot
```

6. *I18n* de un proyecto (II)

- El contenido de un fichero "`.po`" es algo así:

```
# Catalan translations for intlapp package
# Traduccions al catala del paquet <<intlapp>>.
# Copyright (C) 2013 THE vala-i'S COPYRIGHT HOLDER
# This file is distributed under the same license as the intlapp package.
# your name <your.mail@here.is>, 2013.
#
msgid ""
msgstr ""
"Project-Id-Version: intlapp\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2013-10-30 19:59+0100\n"
"PO-Revision-Date: 2013-10-30 20:02+0100\n"
"Last-Translator: name <name@provider>\n"
"Language-Team: Catalan\n"
"Language: ca\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
```

7. *I18n* de un proyecto (III)

```
#: ../src/app.vala:29
msgid "Aplicacion borrada\n"
msgstr "Aplicacio esborrada\n"

#: ../src/app.vala:33
msgid "Aplicacion comenzada\n"
msgstr "Aplicacio comencada\n"

#: ../src/app.vala:25
msgid "Aplicacion creada\n"
msgstr "Aplicacio creada\n"

#: ../src/app.vala:36
msgid "Aplicacion terminada\n"
msgstr "Aplicacio acabada\n"
```

8. *I18n* de un proyecto (IV)

- Una vez traducido, se *compila* para hacer más eficiente su carga: **msgfmt**:

```
msgfmt -c -v -o intlapp.mo ca.po
```

- El significado de las extensiones es éste:

  po
  : *Portable Object*

  pot
  : *Portable Object Template*

  mo
  : *Machine Object*

- Fíjate que los archivos "`.mo`" se llaman *igual* que la aplicación... no como el idioma en el que contienen las cadenas traducidas.

9. *I18n* de un proyecto (V)

- Los archivos ***.mo** se instalarán en "`/usr/share/locale`", en el directorio del idioma correspondiente, y dentro de él, en el directorio "`LC_MESSAGES`".

```
/usr/share/locale/ca/:
LC_MESSAGES

/usr/share/locale/ca@valencia/:
LC_MESSAGES

/usr/share/locale/es_ES/:
LC_MESSAGES
```

- Es decir: `/usr/share/locale/`**`idioma`**`/LC_MESSAGES/`**`app`**`.mo`

10. *I18n* de un proyecto (VI)

- Y en nuestro código... procedemos a (1)*iniciar* ***gettext*** y (2)*marcar las cadenas a traducir*.

```
int main(string[] args) {
     string localedir;

     ////////////////////////////////////
     / Inicio y preparacion para I18N //
     ////////////////////////////////////////////////////////////////////////
     if (Config.DEVELOPMENT_MODE == "ON") { // App not installed,
                                            // catalogs are here
             localedir = "src/po";
     } else {
             localedir = Config.PACKAGE_LOCALE_DIR;
     }
     Intl.setlocale (LocaleCategory.MESSAGES, "");

     Intl.bindtextdomain (Config.GETTEXT_PACKAGE, localedir);
     Intl.bind_textdomain_codeset (Config.GETTEXT_PACKAGE, "UTF-8");
     Intl.textdomain (Config.GETTEXT_PACKAGE);
     ////////////////////////////////////////////////////////////////////////
}
```

11. *I18n* de un proyecto (VII)

- Los valores de la configuración podrían ser estos:

```
namespace Config {
    public const string DEVELOPMENT_MODE    = "ON";
    public const string GETTEXT_PACKAGE     = "intlapp";
    public const string PACKAGE_LOCALE_DIR = "/usr/local/share/locale/";
    public const string PROJECT_DIR = "/home/usuario/proyectos/ejemplo-i18n";
}
```

- En modo desarrollo "`intlapp.mo`" se encuentra en:

```
src
|-- po
    |-- ca
        |-- LC_MESSAGES
            +-- intlapp.mo
```

12. *I18n* de un proyecto (VIII)

- El marcado de cadenas lo hacemos así:

```
class Dca.Application : GLib.Object {
    public Application () {
            stdout.printf ( _("Aplicacion creada\n") );
    }

    ~Application () {
            stdout.printf ( _("Aplicacion borrada\n") );
    }

    public void run () {
            stdout.printf ( _("Aplicacion comenzada\n") );
    }
    public void exit () {
            stdout.printf ( _("Aplicacion terminada\n") );
    }
}
```

- En realidad , ocurre esto: **#define _(x) gettext(x)**

13. *I18n* de un proyecto (IX)

- El mantenimiento de los ficheros "`.po`" lo haremos de este modo:
  - Extraemos el fichero *.pot* como ya hemos visto. . . (*xgettext*)
  - Al fichero *.po* (*catalan.po*) le adjuntamos las modificaciones que haya habido: ***msgmerge -s -U po/catalan.po po/intlapp.pot***

14. Gettext en C, C++

- Debes incluir la cabecera `libintl.h`: *#include <libintl.h>*
- En la llamada a `xgettext` te puede ser útil la opción: *-k_* ← símbolo de subrayado!
- También puedes llamar a `xgettext` con la opción `-a`, la cual extrae todas las cadenas aunque no estén marcadas.
- Si no está definida la *macro* "_", definela tú: `#define _(x) gettext(x)`
- No sólo para `C` o `C++`, si al usar la variable `LANG` probando tu aplicación no cambia el locale para el idioma, prueba con la variable `LANGUAGE`.
- En el ejemplo visto en *Vala*, recuerda que la adaptación de *gettext* a este lenguaje introduce un espacio nombres llamado *Intl*, en lenguajes como `C` o `C++` no existe dicho espacio de nombres, por lo que los identificadores carecen de él:

```
setlocale (LC_MESSAGES, "");
bindtextdomain (GETTEXT_PACKAGE, localedir);
bind_textdomain_codeset (GETTEXT_PACKAGE, "UTF-8");
textdomain (GETTEXT_PACKAGE);
```

15. Edición de ficheros *.po*

- Los ficheros *.po* son archivos de texto. . .
- Se pueden editar con cualquier editor de textos. . .
- Sublime, vim, emacs, eclipse. . .
- Pero también existen herramientas especializadas. . . poedit, gtranslator.

16. Show-time:

- Veamos un ejemplo sencillo de I18N + L10N.
- Idioma original: castellano. Traducimos a catalán.
- Código escrito en vala.

17. Práctica individual:

- Prepara para I18N el código de alguna práctica tuya o algún código que crees en el laboratorio para ello.
- Traduce los mensajes que pueda emitir a varios idiomas (castellano, catalán, inglés, etc...).
- Después de haber creado una primera versión, añade o elimina cadenas para que puedas probar la herramienta `msgmerge`.

ENTREGA:

- La práctica se entregará en pracdlsi en las fechas allí indicadas.

18. Aclaraciones

EN NINGÚN CASO ESTAS TRANSPARENCIAS SON LA BIBLIOGRAFÍA DE LA ASIGNATURA.

- Debes estudiar, aclarar y ampliar los conceptos que en ellas encuentres empleando los enlaces web y bibliografía recomendada que puedes consultar en la página web de la ficha de la asignatura y en la web propia de la asignatura.