

Internacionalización del Software

I18N

DAI-18

L10N (LOCALIZATION) Adaptar el software a otro país con otro:

- Alfabeto
- Idioma de la interfáce
- Formato de fechas
- Moneda
- Formato de las cantidades

I18N (INTERNACIONALIZATION) Preparar los programas para poder localizarlos fácilmente y para que cambien de entorno dinámicamente según la elección del usuario

En general **localizar** consiste en traducir:

- La interface: menús, cuadros de diálogo, plantillas, etc, están en archivos `.resx` (Windows) o `.po` (Linux)
- La ayuda (archivos de hipertexto)
- Resto de la documentación, sitios web, etc, si los hubiera
- Adaptación cultural de la interface (colores, términos inapropiados, etc)

Esto es tarea de traductores

Tecnología para internacionalizar software

LOCALE En UNIX desde 1995

MUI En Windows desde Windows 2000

http://en.wikipedia.org/wiki/Multilingual_User_Interface

Se trata de separar la parte dependiente de la lengua en archivos separados

sin internacionalizar

programa.exe (español)
programa.exe (inglés)
programa.exe (francés)

internacionalizado

programa.exe
mensajes_es.dll
mensajes_fr.dll
...

- 1 El ejecutable es único para todas las lenguas
- 2 Las traducciones están en archivos independientes, hechas por especialistas
- 3 Se puede cambiar de idioma en tiempo de ejecución, dependiendo de la instalación del S.O. (escritorio) o del navegador (web)

Es la librería tradicional de UNIX para **i18N**, la que se usa en linux para los programas de escritorio, y en PHP, Python (**Django**, Flask) etc, para aplicaciones web

Para aplicaciones Java o en Android, la internacionalización se hace de una manera parecida, cambian los formatos de los archivos involucrados

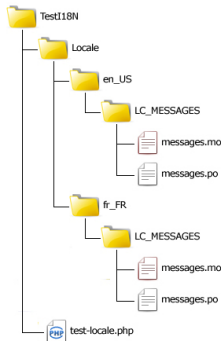
<https://en.wikipedia.org/wiki/Gettext>

<https://docs.python.org/3/library/gettext.html>

<https://docs.djangoproject.com/en/2.1/topics/i18n/>

Unix: gettext, archivos .mo

Los textos para cada lengua, van en archivos `.mo`, (machine object), compilados a partir de los textos en archivos `.po`, (portable object),



que están en un directorio aparte para cada locale (~lengua),
por defecto en `/usr/share/locale`

Nombres de los locales (UNIX)

Para cada entorno se utiliza un 'locale' distinto, cada uno con su nombre:

lengua[_pais][.codificación][@modificación]

es	Español
es_AR	Español para Argentina
es_AR.UTF-8	en codificación utf-8
es_ES@euro	español para España modificación euro

<code>locale</code>	para ver el locale actual
<code>locale -a</code>	para ver todos los locales instalados

LOCALE Cambio del comportamiento de alguna de las funciones de la librería estandar del C, según el valor de las variables de entorno asignadas a cada categoría

Categorías del locale

LC_CTYPE	Relacionada con la codificación (<code>islower()</code> , ...)
LC_COLLATE	Relacionada con ordenamiento (<code>strcoll()</code> , ...)
LC_MESSAGES	Relacionada con la salida (print) (<code>gettext()</code>)
LC_MONETARY	Relacionada con signo de moneda
LC_NUMERIC	Relacionada con formatos de números (<code>printf()</code> , ...)
LC_TIME	Relacionada con formatos de fechas

Variables de entorno

Para decidir el locale en tiempo de ejecución se consultan las variables de entorno (en escritorio):

Primero se consulta **LC_ALL**

luego se consulta **LC_CTYPE**, etc

y si no **LANG**

Para aplicaciones web se consulta la cabecera de http **HTTP_ACCEPT_LANGUAGE** en el request, que manda el navegador

LC_MESSAGES: gettext()

Para cambiar los mensajes:

En lugar de: `printf "Hola mundo\n";`

Se pone: `printf gettext("Hola mundo\n");`

o redefiniendo

la función `gettext` `printf _("Hola mundo\n");`

`gettext` busca el texto correspondiente al locale actual en tiempo de ejecución, en un archivo distinto, por defecto:

`/usr/share/locale/en_US/LC_MESSAGES/mi_programa.mo`

Archivos .po (portable object)

Los archivos '.mo' (machine object, compilados) para cada lengua, se generan a partir de archivos de texto '.po' (portable object)

Por ejemplo, para la traducción al inglés se haría un archivo 'mi_programa.po' para el directorio (y el locale) **en_US**, que contendría líneas como:

```
# Traducciones al inglés
msgid "Hola mundo\n"
msgstr "Hello world\n"
...
```

Se puede usar el programa **xgettext** para sacar una automáticamente una plantilla con todos los mensajes bajo gettext

Archivos .mo (machine object)

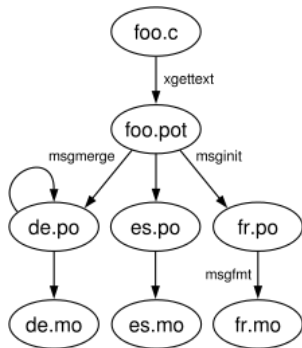
Una vez traducidos, los archivos .po se compilan con **msgfmt** a archivos .mo, y se ponen en los directorios correspondientes a los locales:

```
/usr/share/locale/en_US/LC_MESSAGES/mi_programa.mo
```

EL proceso sería:

- 1 Poner `gettext()` donde se vaya traducir en el código fuente
- 2 Crear la plantilla **.pot** pasando la utilidad `xgettext` al código fuente
- 3 Traducir rellorando una plantilla para cada lengua, poniéndolas en un archivo **.po**
- 4 Compilar las traducciones con la utilidad `msgfmt` a un archivo **.mo**
- 5 Poner las traducciones en sus locales correspondientes al instalar el programa

Archivos implicados



Ejemplo en C

```
#include <stdio.h>
#include <locale.h>
#include <libintl.h>

#define _(str) gettext(str)

int main () {

    setlocale (LC_ALL, "");
    bindtextdomain("test_i18n", "./locale");
    textdomain("test_i18n");

    printf (_("\tEsto es un test\n"));

    salida (_("Fin"));
}
```

Ejemplo en Python

```
#!/usr/bin/env python3

import os, locale, gettext
from gettext import gettext as _

APP_NAME = 'test'
APP_DIR = os.getcwd()

# locale dir, en lugar de /usr/share/locale
LOCALE_DIR = os.path.join(APP_DIR, 'locale')

# uso la lengua local del ordenador
# para cambiarla locale.setlocale(category [,locale])
lengua, codificacion = locale.getlocale()

# buscara las traducciones en 'locale_dir/es_ES/LC_MESSAGES/test.mo'
domain = gettext.textdomain(APP_NAME)
print (domain)

# en directorio
locale_dir = gettext.bindtextdomain(domain, LOCALE_DIR)
print (locale_dir)
```

Ejemplo en Python, continuación

```
print (_('Hello world'))
print (_('canyon'))
print (_('Saved %d files' % (2)))

# Extraemos las plantillas de traducción
# pybabel extract . --output=test.pot

# Creamos los archivos .po en su sitio (directorio 'locale')
# para traducción al español
# pybabel init -D test -d locale -i test.pot -l es

# Traducimos los archivos .po

# Compilar a .mo
# pybabel compile -d locale -l es -D test
```

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2007-02-21 09:53+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: test_il18n.c:20
#, c-format
msgid "\tEsto es un test\n"
msgstr ""

#: test_il18n.c:24
```

Para cada mensaje:

```
white-space
# translator-comments
#. automatic-comments
#: reference...
#, flag...
msgid untranslated-string
msgstr translated-string
```

```
#: test_i18n.c:20
#, c-format
msgid "\tEsto es un test\n"
msgstr "\tThis a test\n"

#: test_i18n.c:24
msgid "Fin"
msgstr "End"
```

A veces los mensajes varían con el número (singular o plural)
p.e.:

```
if (n == 1)
    printf ("%d archivo borrado", n);
else
    printf ("%d archivos borrados", n);
```

Para estos casos se utiliza la función **ngettext**

```
printf (ngettext  
    ("Un archivo", "%d archivos", n), n);
```

El primer argumento es el singular, y se usa como índice en la búsqueda, y el segundo es el plural, y el tercero es el número

En la cabecera del archivo .po se pone la especificación para sacar la forma del plural con sintaxis de C

```
# Español, inglés, etc, 2 formas
Plural-Forms: nplurals=2; plural=(n != 1);

# Francés, 2 formas
Plural-Forms: nplurals=2; plural=(n > 1);

# Polaco
Plural-Forms: nplurals=3;
    plural=(n==1 ? 0 :
            (n%10>=2 && n%10<=4) &&
            (n%100<10 or n%100>=20)
            ? 1 : 2);
```

Y en el contenido del archivo .po

```
msgid "Un archivo"  
msgid_plural "%d archivos"  
msgstr[0] ""  
msgstr[1] ""  
.. etc
```

Hay programas para facilitar la traducción de los archivos .po:

poedit, lokalize, gtranslator, virtaal

o con:

http://en.wikipedia.org/wiki/Google_Translator_Toolkit

Con estos programas se hace una traducción semi-automática, (asistida) con:

- 'Memorias de Traducción' (traducciones anteriores)
- Conexión a APIs de Traducción automática (google, bing)
- Glosarios especializados
- Avisos de errores de ortografía, puntuación, etc

<http://www.microsoft.com/language/es-es/default.aspx>

En el manual de Django

<https://docs.djangoproject.com/en/2.1/topics/i18n/translation/>
se detalla todo,

en síntesis:

```
from django.utils.translation import ugettext as _  
  
x = _('para ser traducido')
```

o en los templates

```
{% load i18n %}  
  
{% trans "Esto se traducirá" %}
```

Hay que crear un directorio **locale** donde está 'manage.py'

y

```
django-admin makemessages -l en
```

Traducir y

```
django-admin compilemessages -l en
```

y añadir en el middleware, al final

```
'django.middleware.locale.LocaleMiddleware',
```