Gettext

Gettext[1] è un sistema che aiuta nella traduzione dei messaggi dei programmi e al loro mantenimento, il quale però non fa parte della libreria standard dei linguaggi di programmazione. Ci possono essere molti modi per realizzare un programma multilingua, ma Gettext rappresenta probabilmente il metodo più semplice in pratica, consentendo la traduzione successiva senza interferire con un eseguibile già pronto, purché predisposto per questo.

71.1 Principio di funzionamento

La logica di Gettext è molto semplice: il programma incorpora solo i messaggi in inglese; all'esterno si associano dei file, uno per ogni linguaggio disponibile, con le traduzioni corrispondenti. Non è necessario «codificare» i messaggi in qualche modo, perché la corrispondenza avviene in modo letterale, in base al testo originale.

```
msgid "%s: cannot create the temporary file %s\n"
msgstr "%s: non è possibile creare il file temporaneo %s\n"
```

L'esempio, che mostra un estratto ipotetico di un file PO di Gettext (*Portable object*), serve a comprendere il concetto: la stringa preceduta dalla parola chiave '**`msgid`**' (*message identity*) è quella di riferimento, che viene rimpiazzata automaticamente da quella sottostante, preceduta dalla parola chiave '**`msgstr`**'.

Le stringhe e le traduzioni di Gettext sono costanti, nel senso che '**`%s`**' viene preso come tale, mentre è il programma che lo sostituisce opportunamente. In questo senso, bisogna considerare che Gettext è nato per il linguaggio C, per essere usato in stringhe che siano argomento di funzioni come ***printf()*** e ***sprintf()***.

71.2 Fasi di preparazione

«

La predisposizione di un programma per Gettext potrebbe essere fatta in modo più o meno automatico, attraverso strumenti specifici, oppure si può procedere in modo più semplice, anche se più oneroso dal punto di vista del tempo impiegato. Qui si intende mostrare questo modo più semplice per permettere al lettore di comprendere il concetto. La documentazione di Gettext è di per sé molto dettagliata.

Per prima cosa, il sorgente C deve essere predisposto attraverso l'inclusione di alcuni file di intestazione, quindi le stringhe vengono inglobate dalla funzione ***gettext()***. Quello che segue è il classico programma che visualizza un messaggio ed esce; si suppone che si tratti del file '`ciao.c`':

```
#include <stdio.h>
int main ()
{
    printf ("Hello world\n");
}
```

Nel listato successivo si vede come deve essere trasformato; va osservata l'inclusione del file di intestazione '`libintl.h`':

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

#define PACKAGE "ciao"
#define LOCALEDIR "/var/tmp"

int main ()
{
    setlocale (LC_ALL, "");
    bindtextdomain (PACKAGE, LOCALEDIR);
    textdomain (PACKAGE);

    printf (gettext ("Hello world\n"));
}
```

Le funzioni '**bindtextdomain**' e '**textdomain**' utilizzano come argomenti delle macro (costanti manifeste), in modo da generalizzare il funzionamento e rendere esterna la definizione di queste componenti. A parte questi particolari, si nota che ***printf()*** non ha più come argomento la costante di prima, ma la funzione ***gettext()***.

Il programma può essere compilato, anche se non è ancora disponibile una traduzione:

```
$ cc -o ciao ciao.c[Invio]
```

La fase successiva richiede la creazione di un file PO, attraverso l'aiuto del programma '**xgettext**':

```
$ xgettext ciao.c[Invio]
```

Quello che si ottiene nella directory corrente è il file '`messages.po`', contenente esattamente il testo seguente:

```
# SOME DESCRIPTIVE TITLE.
```

```
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2000-05-15 23:05+0200\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: ENCODING\n"

#: ciao.c:18
msgid "Hello world\n"
msgstr ""
```

Questo file deve essere modificato, in particolare per ciò che riguarda le prime direttive, oltre che per aggiungere la traduzione della frase che viene visualizzata dal programma. Per esempio, così:

```
# Ciaomondo PO file.
# Copyright (C) 2000 Pinco Pallino
# Pinco Pallino <ppinco@dinkel.brot.dg>, 2000.
#
msgid ""
msgstr ""
"Project-Id-Version: ciao-0.1\n"
"POT-Creation-Date: 2000-05-15 23:05+0200\n"
"PO-Revision-Date: 2000-05-15 22:52+0200\n"
"Last-Translator: Pinco Pallino <ppinco@dinkel.brot.dg>\n"
"Language-Team: Italian <it@li.org>\n"
"MIME-Version: 1.0\n"
```

```
"Content-Type: text/plain; charset=iso-8859-1\n"
"Content-Transfer-Encoding: 8bit\n"

#: ciao.c:18
msgid "Hello world\n"
msgstr "Ciao mondo\n"
```

Si osservi che è stato necessario togliere la riga contenente il commento speciale '**#, fuzzy**'.

Il file viene salvato con un nome appropriato; per esempio '`ciao.po`'. Quindi si passa alla sua compilazione, per ottenere il file '`ciao.mo`':

```
$ msgfmt -vvvv -o ciao.mo ciao.po[Invio]
```

Dal momento che il file in questione contiene la traduzione in italiano del programma, deve essere collocato all'interno della gerarchia '`it/LC_MESSAGES/`', a sua volta a partire dalla directory dichiarata con la funzione ***bindtextdomain()***, cioè '`/var/tmp/it/LC_MESSAGES/`' secondo quanto definito nel sorgente C.

A questo punto, dopo la collocazione appropriata del file compilato della traduzione, se la configurazione locale è corretta, lanciando l'eseguibile '**ciao**' si dovrebbe vedere il messaggio tradotto. Eventualmente si veda quanto descritto nella sezione 16.11 per quanto riguarda la configurazione della localizzazione.

71.3 Abbinamento a un «pacchetto»

«

Perché Gettext sappia qual è il file che contiene i messaggi tradotti, nell'ambito della configurazione locale, fa riferimento a un nome che viene definito «pacchetto», che di solito si sceglie opportunamente simile a quello del programma per il quale si fa la traduzione:

```
textdomain ("pippo");
```

L'esempio mostra l'istruzione da usare in un programma C per stabilire il nome del pacchetto secondo Gettext. Questo nome stabilisce che Gettext debba cercare il file '***sigla_locale***`/LC_MESSAGES/pippo.mo`'. Gettext determina il nome della prima parte del percorso, corrispondente a ciò che qui è stato mostrato con la metavariabile ***sigla_locale***, analizzando alcune variabili di ambiente; precisamente segue questo ordine:

- '**`LANGUAGE`**'
- '**`LC_ALL`**'
- altre variabili '**`LC_`***'
- '**`LANG`**'

Dal valore contenuto in queste variabili si estrae la prima parte: quella che arriva fino al primo punto, se c'è. In pratica, se per ipotesi la variabile '**`LANG`**' contiene il valore '**`it_IT.ISO-8859-1`**', per Gettext è importante solo '**`it_IT`**'. Tuttavia, anche questa informazione tende a essere eccessiva, dal momento che contiene, oltre al linguaggio, anche l'area nazionale. In pratica, alla fine contano solo le prime due lettere, che esprimono il linguaggio in base allo standard ISO 639 (tabella 13.4).

Gettext analizza il contenuto delle variabili di ambiente perché con la funzione ***setlocale()*** è stata azzerata internamente la definizione '**LC_ALL**'. Usando la funzione ***setlocale()*** si potrebbe imporre un certo linguaggio, indipendentemente dalle variabili di ambiente relative.

Tornando all'esempio iniziale, si tratta del file '`it/LC_MESSAGES/pippo.mo`', che in condizioni normali, Gettext cerca a partire dalla directory '`/usr/share/locale/`' (o eventualmente da un'altra posizione in base al modo in cui è stato compilato). Tuttavia è possibile richiedere espressamente una collocazione differente attraverso un'istruzione già vista da collocare nel programma interessato:

```
bindtextdomain ("pippo", "/var/tmp");
```

In tal caso, se si scrive questo in un programma, Gettext va a cercare precisamente il file '`/var/tmp/it/LC_MESSAGES/pippo.mo`'.

71.4 Creazione e mantenimento dei file PO

È già stato mostrato in breve come si crea un file PO attraverso il programma '**xgettext**'. È il caso di osservare che '**xgettext**' può ricevere l'indicazione di più file sorgenti che fanno capo allo stesso dominio di traduzione:

```
xgettext [opzioni] file_sorgente...
```

In particolare, tra le opzioni può essere interessante segnalare '**--default-domain=*dominio***', che serve a '**xgettext**' per conoscere il dominio a cui si fa riferimento, creando così il file '***dominio*.po**', invece del solito '`messages.po`'.

```
$ xgettext --default-domain=ciao *.c[Invio]
```

L'esempio mostra come ottenere il file 'ciao.po' a partire da tutti i file che terminano con l'estensione '.c'.

Quando si aggiornano i sorgenti di un programma già tradotto, si pone il problema di aggiornare nello stesso modo i file PO precedenti. Per fare questo si deve ricreare il file PO iniziale non tradotto, nel modo appena visto, quindi si usa il programma '**msgmerge**':

```
msgmerge [opzioni] file_po_originale file_po_successivo > file_po_aggiornato
```

In pratica, '**msgmerge**' fonde assieme due file PO, preservando le traduzioni del primo file riferite a messaggi che si trovano ancora nel secondo. Per esempio, se si dispone già del file 'vecchio.po' con le traduzioni, mentre con '**xgettext**' è appena stato generato un file PO non tradotto per lo stesso programma, che qui viene chiamato 'non_tradotto.po', si può ottenere un nuovo file PO con le traduzioni vecchie ancora valide e con i messaggi nuovi da tradurre:

```
$ msgmerge vecchio.po non_tradotto.po > nuovo.po[Invio]
```

Naturalmente, questa operazione si fa nel momento in cui ci si accinge ad aggiornare materialmente la traduzione del programma, altrimenti questo lavoro non avrebbe senso, dal momento che un file PO contenente messaggi non tradotti non può essere compilato.

```
msgfmt [opzioni] file_po
```

Il programma '**msgfmt**' è quello che si occupa di compilare i file PO

ottenendo i file MO, adatti alla propria piattaforma. È praticamente indispensabile utilizzare l'opzione '**--output-file=*file_mo***' ('**-o**'), per indicare il nome del file da creare. Inoltre, è opportuno utilizzare più di una volta l'opzione '**--verbose**' ('**-v**') per avere una visione chiara del procedimento, ovvero dei motivi per i quali alle volte il file non viene compilato.

```
$ msgfmt -vvvv --output-file=prova.mo prova.po[Invio]
```

L'esempio mostra l'utilizzo tipico di questo programma, dove in particolare viene richiesto un livello di dettaglio delle informazioni generate molto elevato (quattro volte '**-v**').

71.4.1 Commenti «fuzzy»

Ogni volta che qualche indicazione all'interno di un file PO è incerta, in quanto predefinita o determinata automaticamente in modo non sicuro, viene aggiunto un commento speciale contenente la parola '**fuzzy**'. In presenza di commenti del genere si richiede un intervento manuale, dopo il quale deve essere rimossa tale parola, altrimenti '**msgfmt**' si rifiuta di completare la compilazione dei file PO.

71.5 Gettext con i programmi Perl

Esiste la possibilità di utilizzare Gettext anche nei programmi Perl. Per questo è necessario includere nel programma Perl il riferimento a un modulo esterno: Perl-gettext. Il tutto si svolge in maniera molto simile a un programma C, inserendo inizialmente le istruzioni seguenti:

```
use POSIX;
use Locale::gettext;
setlocale (LC_ALL, "");
textdomain ("dominio_gettext");
[bindtextdomain ("dominio_gettext", "directory");]
```

Per esempio, se si tratta del programma «Pippo», il dominio per Gettext potrebbe essere convenientemente «pippo», arrivando al risultato seguente:

```
use POSIX;
use Locale::gettext;
setlocale (LC_ALL, "");
textdomain ("pippo");
bindtextdomain ("pippo", "/opt/pippo/locale");
```

Potrebbe essere che la funzione ***bindtextdomain()*** non si comporti come previsto; in tal caso sarebbe meglio evitarne l'uso.

Per il resto, tutto funziona come per i sorgenti scritti in C:

```
print STDOUT (gettext ("Hello world\n"));
```

Tuttavia, Perl non è identico al C, per cui occorre osservare alcune situazioni specifiche. In particolare, non è possibile inserire in un argomento della funzione ***gettext()*** una variabile di Perl che deve essere espansa, perché questa espansione avverrebbe prima che ***gettext()*** possa ricevere tale argomento. Pertanto, l'esempio seguente non può essere tradotto:

```
# Esempio errato.
print STDOUT (gettext ("Il file $file contiene "
                       "caratteri non validi\n"));
```

Il modo giusto di agire è quello di sostituire ***print()*** con ***printf()***, come nell'esempio seguente:

```
# Esempio corretto.
printf STDOUT (gettext ("Il file %s contiene "
                        "caratteri non validi\n"),
               $file);
```

Infatti, il parametro '**%s**' viene sostituito alla fine da '**printf**', per cui inizialmente la stringa non viene modificata.

Un altro problema da considerare sono i messaggi lunghi, che richiedono più righe. In Perl si potrebbe fare una cosa del genere:

```
printf STDOUT
  (gettext
    ( "Usage: %s --input-type=TYPE INPUT_FILE REPORT_FILE\n"
    . "       %s --help\n"
    . "       %s --version\n"
    . "\n"
    . "Check for HTTP and FTP URI inside a text.\n"
    . "\n"
    . "Options:\n"
    . "--help              display this help and exit.\n"
    . "--version           display version information \n"
    . "                      and exit.\n"
    . "--input-type=TYPE   define the input type:\n"
    . "    standard        input is a simple text file;\n"
    . "    html, sgml      input is a typical SGML file;\n"
    . "    texi, texinfo   input is a Texinfo source \n"
    . "                      file.\n"
    . "\n"
    . "Arguments:\n"
    . "\n"
    . "INPUT_FILE          the input file.\n"
    . "\n"
    . "REPORT_FILE         a file that is generated with \n"
```

```
    . "                          the reported errors.\n"),
    $program_name, $program_name, $program_name);
```

Ma questo non viene riconosciuto da '**xgettext**' che riesce a prelevare solo la prima riga:

```
#: urichk:55
#, c-format
msgid "Usage: %s --input-type=TYPE INPUT_FILE REPORT_FILE\n"
msgstr ""
```

In queste situazioni eccezionali, occorre intervenire a mano nel file PO; sia la prima volta che si crea il file, sia tutte le volte successive in cui lo si aggiorna.

71.5.1 Adattare il sorgente Perl per facilitare l'estrazione dei messaggi da tradurre

«

Se si prendono delle piccole precauzioni nella scrittura delle stringhe con Perl, è possibile filtrare il sorgente successivamente, per passarlo a '**xgettext**' in modo che questo possa interpretarlo correttamente (come se fosse un programma C). In pratica, occorre fare in modo che le stringhe siano unite, senza ottenerle attraverso un concatenamento. L'esempio già mostrato va modificato nel modo seguente:

```
    printf STDOUT
      (gettext
        ("\
Usage: %s --input-type=TYPE INPUT_FILE REPORT_FILE\
       %s --help\
       %s --version\
\
Check for HTTP and FTP URI inside a text.\
```

```
\
Options:\
--help                display this help and exit.\
--version             display version information and exit.\
--input-type=TYPE     define the input type:\
    standard          input is a simple text file;\
    html, sgml        input is a typical SGML file;\
    texi, texinfo     input is a Texinfo source file.\
\
Arguments:\
\
INPUT_FILE            the input file.\
\
REPORT_FILE           a file that is generated with the \
                        reported errors."),
        $program_name, $program_name, $program_name);
```

A questo punto, si può realizzare un piccolo programma Perl che inserisce il codice '**\n**' alla fine delle righe (sostituendolo alla barra obliqua inversa) e sostituisce il codice '**\@**' con '**@**':

```
#!/usr/bin/perl
$line = "";
while ($line = <STDIN>)
  {
    $line =~ s/\\$/\\n\\/;
    $line =~ s/\\\@/\@/g;
    print STDOUT ($line);
  }
```

Come si vede, il programma Perl che si ottiene legge dallo standard input ed emette il risultato della sua trasformazione attraverso lo standard output.

71.5.2 Alleviare gli inconvenienti di un modulo in più

«

Scrivere un programma Perl che faccia uso di Gettext, significa costringere a installare il modulo Perl-gettext. Purtroppo, una delle cose che complicano di più l'utilizzo di programmi Perl sono i moduli aggiuntivi necessari che devono essere installati perfettamente come previsto.

Questo potrebbe sembrare un problema secondario; invece non lo è affatto. A questo punto, se si vuole consentire al proprio programma Perl di funzionare anche in un ambiente non tanto amichevole, si deve prevedere una via di uscita:

```
#!/usr/bin/perl
#
…

use POSIX;
use Locale::gettext;
setlocale (LC_ALL, "");
textdomain ("pippo");

#sub gettext
#{
#    return $_[0];
#}

…
```

Come si vede nell'esempio, appare la dichiarazione di una funzione commentata, il cui scopo sarebbe quello di sostituirsi alla funzione ***gettext()*** del modulo '**`Locale::gettext`**'. Se non si dispone di Perl-gettext basta commentare la prima parte e togliere i commenti

dalla seconda: ovviamente i messaggi rimangono così nella lingua di partenza.

```
#!/usr/bin/perl
#
...

#use POSIX;
#use Locale::gettext;
#setlocale (LC_ALL, "");
#textdomain ("pippo");

sub gettext
{
    return $_[0];
}

...
```

[1] **Gettext** GNU GPL