

# **Internationalisierung von GNU Softwareprojekten**

**Titze Stefan**

**05.07.2004**

# 1. Internationalisierung

## 1.1 Einleitung

Bei GNU Projekten kann jeder mitmachen und einen Teil der Entwicklungsarbeit übernehmen. Möglich wird dies durch unsere „vernetzte Welt“, das Internet. Dieses Netzwerk überwindet geographische, chronologische und kulturelle Differenzen zwischen Menschen und ermöglicht so die Bildung von „virtuellen Teams“ zur Entwicklung von GNU Projekten. In diesen „virtuellen Teams“ befinden sich meist Menschen, die allesamt eine andere Sprache sprechen. Meist beherrschen die Entwickler eines Teams eine gemeinsame Sprache mit der sie untereinander kommunizieren und sich austauschen können. In dieser gemeinsamen „Teamsprache“ entstehen meist die ersten Versionen der gemeinsam entwickelten Software. Da GNU Programme unter der GPL stehen und somit von jedem eingesetzt werden dürfen ist es sehr wahrscheinlich, daß die entwickelte Software auch von Benutzern eingesetzt werden will, die der „Teamsprache“ nicht mächtig sind. Es entsteht somit ein Bedarf die entwickelte Software in die gängigsten Sprachen zu übersetzen, und es somit möglichst vielen potentiellen Benutzern zu ermöglichen die Software einzusetzen. Da im Team mehrere Sprachen beherrscht werden --- „virtuelles Team“ --- liegt es nahe schon in einem frühen Entwicklungsstadium der Software an die Internationalisierung zu denken, zumal das Potential für einige Übersetzungen bereits im „virtuellen Team“ gefunden werden kann.

In diesem Dokument werde ich anhand eines kleinen Beispielprojekts die Internationalisierung in GNU Projekten aufzeigen und sie detailliert beschreiben. Die Beschreibung erfolgt anhand des GNU Translation Projects. Das Translation Project hat sich mit der Internationalisierung von Anwendungen auseinandergesetzt und ein Verfahren sowie Tools entwickelt mit denen Software Internationalisierung sehr einfach ermöglicht werden kann.

## 1.2 Erklärungen anhand eines Beispielprojekts

Nachfolgend werde ich die Internationalisierung anhand eines einfachen Beispielprojekts zeigen und erklären. Das Beispiel wird mit dem GNU üblichen Verfahren als Sourcecodepaket unter Verwendung der Autotools erstellt. Da ich das Beispiel anhand des Sourcecodes erkläre und ältere Autotools beim Nachbau aufgrund fehlender Makros mehr Frust als Lust verursachen, sollten folgende Pakete auf dem System vorhanden sein:

- GNU Autoconf --- Version 2.59
- GNU Automake --- Version 1.9.5

- GNU Gettext --- Version 0.14.4

Die Versionsnummern sind hier als Mindestversionsnummern anzusehen. Ich selbst habe das Projekt mit den derzeit neuesten Versionen (Stand 07.07.2005) getestet sowie feststellen müssen, daß Autoconf v2.57 das Projekt nicht korrekt handhabt.

Doch nun zum Beispiel.

## 1.2.1 Schritte zur Internationalisierung

Um ein internationalisiertes Projekt zu erhalten befolgt man als Entwickler am besten nachfolgende Schritte:

- Erster Prototyp ohne Internationalisierung
- Einbau der Internationalisierung in die Buildkette (Autotool)
- Vorbereiten des GNU Sourcepaketes auf die Internationalisierung
- Einbau der Internationalisierung in den Sourcecode
- Schrittweises Übersetzen des Projektes in mehrere Sprachen

Doch zunächst noch einige Worte vorweg. Um ein Programm zu internationalisieren kann verschieden vorgegangen werden. Man könnte z.B. den Sourcecode direkt übersetzen und dann kompilieren. Jedoch müßten dann Sourcecodedateien für jede Sprache existieren. Das kompilierte Programm könnte dann selbstverständlich seine Sprache zur Laufzeit nicht mehr ändern. Außerdem müßten bei einer Sourcecodeänderung alle Versionen der betroffenen Sourcecodedateien ebenfalls abgeändert werden. Ist leicht verständlich warum diese Vorgehensweise nicht benutzt wird. Sie würde zu Inkonsistenzen zwischen verschiedenen Sprachversionen desselben Programms führen, z.B. wenn vergessen wurde einen Fehler aus dem deutschen Sourcecode zu entfernen, so hätte nur die deutsche Version diesen Fehler und alle anderen Versionen nicht. Darüber hinaus müßten auch für jede Sprachversion eigene Patches erstellt werden. Insgesamt resultiert aus dieser Vorgehensweise ein beachtlicher Mehraufwand an Organisation und Entwickeln, so daß nicht dermaßen vorgegangen wird.

Doch wie wird dann internationalisiert? Das Problem der ersten Variante ist, daß für jede Sprache eine Version einer Sourcecodedatei vorhanden ist. Günstiger wäre es, wenn nur eine einzige Version vorhanden ist. So muß nur eine Version auf Fehler gepflegt werden. Kurzum für jeden String im Sourcecode, der übersetzt werden soll müßte das Programm z.B. in einer Tabelle nach der Sprache sehen, die derzeit ausgewählt ist und diesen String dann anzeigen. Genau so funktioniert die Internationalisierung. Die Sprache, die ausgewählt ist befindet sich dabei in der Systemvariablen „LANG“. Debian 3.1, welches mit der Standardsprache

deutsch installiert wurde, gibt auf den Befehl „echo \$LANG“ die Zeichenkette „de\_DE@euro“ aus. Das erste Kürzel „de“ steht für die Sprache Deutsch, das zweite Kürzel „DE“ steht für das Land Deutschland und das zusätzliche „@euro“ zeigt an, daß ein Zeichensatz mit dem Eurosymbol verwendet wird. Die Funktionalität der Suche nach einem String in einer Art Übersetzungstabelle funktioniert hierbei automatisch mit der „gettext“ Funktion. So wird z.B. nach dem String „Hello“ gesucht wenn folgende Befehlszeile abgearbeitet wird „printf (gettext ("Hello"))“. Pro Sprache existiert eine eigene Tabelle. Jede Tabelle einer Sprache entsteht aus einem Master, dem sogenannten „Portable Object Template“ (pot). Soll eine bestimmte Sprachübersetzung hinzugefügt werden, so wird aus dem Master eine sprachabhängige Datei, das sogenannte „Portable Object“ (po). In die „po“ Datei werden die übersetzten Strings zu den jeweiligen Referenzstrings des Sourcecodes eingetragen. Die sprachabhängigen Tabellen müssen anschließend noch kompiliert werden und stehen dann als „Machine Object“ (mo) zur Verfügung.

## 1.2.2 Erster Prototyp

Nachfolgend stelle ich den Sourcecode für den ersten Prototyp vor. Bei dem Programm handelt es sich um das berühmte „Hello, World!“ Programm. Das Programm ist in der Programmiersprache „C“ geschrieben. Weitere Erklärungen zum Sourcecode sind nicht notwendig, da es sich ja um ein triviales Beispiel handelt. Das Beispielprojekt ist in mehrere Unterverzeichnisse aufgeteilt. Das Verzeichnis „./src“ enthält den Sourcecode des Programms; das Verzeichnis „./po“ enthält die übersetzten Programmteile, die ich später noch vorstellen werde. Im Basisverzeichnis „.“ befindet sich die Autoconfdatei „configure.ac“ mit deren Hilfe das „configure“ Skript erstellt wird. Darüber hinaus enthält das Basisverzeichnis die Datei „Makefile.am“ und das „./src“ Unterverzeichnis enthält ebenfalls eine „Makefile.am“ Datei. Diese beiden Dateien werden von den Autotools zur Erstellung der „Makefile.in“ Dateien verwendet; aus diesen werden später beim Konfigurieren des Sourcepaketes die beiden Makefiles „./Makefile“ und „./src/Makefile“ generiert. Die Handhabung der GNU Buildkette werde ich hier an dieser Stelle jedoch nicht weiter ausführen, da diese Arbeit sich mit der Internationalisierung beschäftigt.

```
“./src/main.c”
```

```
#include "config.h"  
#include <stdio.h>
```

```
int main ()  
{  
    printf ("Hello, World!\n\0");
```

```

        return 0;
    }

"./configure.ac"

AC_INIT([hello], [0.1], [bug-report@address.de])
AM_INIT_AUTOMAKE([-Wall -Werror foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT

"./Makefile.am"

SUBDIRS = src

"./src/Makefile.am"

bin_PROGRAMS = hello
hello_SOURCES = main.c

```

### 1.2.3 Anpassung der Autotool Konfigurationsdateien

Nun werde ich kurz erläutern wie unser Beispielprojekt internationalisiert wird. Zunächst müssen einige Änderungen an den Autotooldateien durchgeführt werden, im Anschluß daran muß das Projekt für die Internationalisierung vorbereitet werden (mithilfe eines Tools) und schließlich müssen noch einige kleinere Änderungen an den Autotool Dateien erfolgen.

Zunächst zur „./configure.ac“. Um im „configure“ Skript das Vorhandensein des „gettext“ Packages prüfen zu können muß das Makro „AM\_GNU\_GETTEXT\_VERSION([0.14.4])“ und das Makro „AM\_GNU\_GETTEXT([external])“ in die Konfigurationsdatei eingefügt werden. Das erste Makro veranlaßt das „configure“ Skript nach einer exakten Version (hier 0.14.4) auf dem Zielsystem zu suchen. Normalerweise befindet sich die Funktionalität von gettext in der Bibliothek „libc“. Normalerweise bedeutet, daß die „libc“ gettext bei Linux Systemen beinhaltet. Bei anderen Betriebssystemen wird gettext meist in einer separaten Bibliothek mitgeliefert. Das zweite Makro beschreibt wie nach gettext gesucht werden soll. „external“ bedeutet, daß gettext sowohl in der „libc“ als auch in anderen Bibliotheken gesucht wird. Wird gettext nicht gefunden, so wird die Internationalisierung nicht übernommen. Das Programm ist nach dem Kompilieren nur einsprachig. Die gettext Funktionalitäten wird für die Internationalisierung wie bereits angesprochen benötigt.

Somit ergibt sich nun folgende „configure.ac“

```
AC_INIT([hello], [0.1], [bug-report@address.de])
AM_INIT_AUTOMAKE([-Wall -Werror foreign])
AM_GNU_GETTEXT_VERSION([0.14.4])
AM_GNU_GETTEXT([external])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

## 1.2.4 Vorbereitung des GNU Sourcepaketes

Die Sprachdateien befinden sich laut GNU Konvention im Verzeichnis „./po“. Um „po“ Dateien kompilieren zu können müssen einige spezielle GNU Tools verwendet werden, die allesamt im „gettext“ Packet enthalten sind. Da der Buildprozeß von „mo“ Dateien immer der gleiche ist existiert das Tool „gettextize“. Mit diesem Tool wird die „./po“ Verzeichnisstruktur und einige Konfigurationsdateien für die Autotools im „./po“ Verzeichnis automatisch erstellt. Darüber hinaus wird „configure.ac“ und „./Makefile.am“ angepaßt. In der „configure.ac“ wird die Erstellung einer „./po/Makefile.in“ Datei eingetragen und in die „./Makefile.am“ wird das Unterverzeichnis „po“ und weitere Makros für das internationalisierte Compilieren eingetragen.

Mit dem Aufruf „./gettextize --copy --no-changelog“ wird unser Beispielprojekt nun auf die Internationalisierung vorbereitet. Zusätzlich muß allerdings die Headerdatei „2gettext.h“ ins Projektverzeichnis kopiert werden („./src“). In dieser Headerdatei befinden sich die benötigten Definitionen für die später benutzten Funktionen „gettext“, „setlocale“ und „bindtextdomain“. Die Datei ist im „gettext“ Packet enthalten und befindet sich nach dessen Installation meist im Verzeichnis „/usr/share/gettext“ oder bei manueller Installation im Verzeichnis „/usr/local/share/gettext“.

Da nun diese Headerdatei ebenfalls mit dem Packet ausgeliefert wird muß die Headerdatei noch in die Datei „./src/Makefile.am“ eingetragen werden. Die Variable „\$(LOCALEDIR)“ wird für die Initialisierung der Internationalisierung im Sourcecode benötigt und wird daher vom Makefile an den Sourcecode über die Compilerflags weitergereicht. Mithilfe von „\$(LOCALEDIR)“ können später die „mo“ Dateien für unser Beispielprogramm gefunden werden (meist in „/usr/local/share/locale“). Für die „gettext“ Funktionen wird wie schon erwähnt eine Bibliothek verwendet. Da der Name für diese Bibliothek nicht bei jedem Betriebssystem gleich ist wird hierfür der abstrakte Name „LIBINTL“ in der „./Makefile.am“ als zusätzliche Abhängigkeit eingetragen.

```
“./src/Makefile.am”
```

```
AM_CPPFLAGS = -DLOCALEDIR=\"$(localedir)\"  
bin_PROGRAMS = hello  
hello_SOURCE = main.c gettext.h  
LDADD = $(LIBINTL)
```

Als nächstes muß noch die Datei „./po/Makevars.template” in „./po/Makevars” umbenannt werden. In dieser Datei sind weitere Autotoolmakros für die Internationalisierung enthalten. Besonders interessant ist hierbei das Makro „COPYRIGHT HOLDER” hier kann der Name des Erstellers des Projekts eingetragen werden (in die ./po/Makevars).

Die Datei „./po/POTFILES.in” enthält eine Liste aller Sourcecodeteile für die später eine Übersetzung vorhanden sein soll. Deshalb tragen wir hier unsere Datei „./src/main.c” ein.

```
“./po/POTFILES.in”
```

```
src/main.c
```

## 1.2.5 Internationalisierung des Sourcecodes

Wie bereits vorher erwähnt muß jeder String, der später übersetzt werden soll in die „gettext” Funktion eingepackt werden. Aber damit das Programm später beim Ausführen weiß welche Sprache der Anwender gewählt hat (also in der Systemvariablen \$LANG) muß diese zuerst vom Programm übernommen werden. Hierfür wird die Funktion setlocale verwendet. Mit der „bindtextdomain” Funktion wird der Name der „mo” Datei angegeben, die für die Übersetzung vom Programm verwendet wird, sowie das Basisverzeichnis (\$LOCALEDIR) in dem sich die „mo” Dateien befinden. Mit der Funktion „textdomain” wird die Textdomäne festgelegt. Diese wird immer dann interessant wenn sich in einem Paket mehrere Programme befinden. Sie ist nämlich dann von Programm zu Programm verschieden. In unserem Fall wird hier der Name unseres Paketes übergeben. Schließlich werden die Header „gettext.h” und „locale.h” noch inkludiert. Die Headerdatei „gettext.h” habe ich bereits angesprochen und die bei der Datei „locale.h” handelt es sich um eine ANSI C Headerdatei, in der Standardfunktionalitäten für die Lokalisierung eines Programmes enthalten sind.

```
“./src/main.c”
```

```
#include "config.h"  
#include <locale.h>
```

```

#include "gettext.h"

#define _(String) gettext(String)

int main ()
{
    setlocale (LC_ALL, "");
    bindtextdomain (PACKAGE, LOCALEDIR);
    textdomain (PACKAGE);
    printf (_("Hello, World!\n\n"));
    return 0;
}

```

Wie hier zu sehen ist wird der Aufruf von „gettext (...)“ durch „\_(...)“ substituiert. Diese Abkürzung ist in den meisten internationalisierten GNU Projekten zu finden, denn für jeden String, der später vom Programm in mehreren Sprache ausgegeben werden soll, müsste der Entwickler immer den langen „gettext“ Funktionsaufruf verwenden. Da Entwickler aber meist etwas schreibfaul sind wird statt dessen gerne „\_(...)“ verwendet.

Mit dem Aufruf „./autoreconf --install && ./configure && make“ wird das Sourcecodepaket auf die Internationalisierung vorbereitet. Die „pot“ Datei wird durch diesen Aufruf erstellt („./po/hello.pot“).

## 1.2.6 Übersetzung des Programms

Nachdem alle Schritte durchgeführt worden sind werde ich nun die Übersetzung des Beispielprogramms ins Deutsche erläutern. Das Programm besitzt die Basissprache Englisch. Diese wird immer dann ausgegeben, wenn der „National Language Support“ (NLS) nicht vom Zielcomputer unterstützt wird. Dies ist z.B. der Fall wenn die gettext Funktionalitäten auf dem Zielcomputer nicht gefunden werden oder wenn dem Konfigurationsskript „configure“ der Parameter „--disable-nls“ übergeben wird. Ansonsten zeigt das Programm die Sprache an, die in der „\$LANG“ Systemvariablen eingestellt ist oder die Variable „LANG“ wird vor Programmstart gesetzt

```
./LANG=de_DE hello
```

Und nun zur Übersetzung. Wir haben bereits eine „pot“ Datei. Diese Datei ist unser Master für die Übersetzungen. In diesem Master befinden sich genau die Strings, die mit dem „gettext“ Funktionsaufruf markiert wurden. Diese Strings sind im Feld „msgid“ in der „pot“ Datei zu finden. Für die Übersetzung müssen wird lediglich das dazugehörige Feld „msgstr“ übersetzt ausfüllen.

Nun fügen wir eine deutsche Übersetzung des Programms hinzu. Um eine „leere“, d.h. nicht übersetzte „po“ Datei zu erhalten gibt es das Tool „msginit“ (im „gettext“ Packet). Mit dem Aufruf „./po/msginit -l de“ wird eine „leere“ deutsche „po“ Datei erzeugt. In dieser Datei wird nun des Feld „msgid \"Hello, World!\n““ gesucht und im darunterliegenden Feld „msgstr “““ wird der übersetzte String eingetragen („msgstr \"Hallo, Welt!\n““). Im Anschluß muß unsere neue Sprache des Programms noch in die Datei „./po/LINGUAS“ eingetragen werden. Diese Datei enthält eine einzeilige Auflistung aller Sprache, in die das Programm übersetzt wurde (die Sprachen sind durch einen Leerschritt voneinander getrennt). Wir tragen also nun „de“ in die Datei ein (Falls sie nicht existiert, so wird sie einfach erstellt).

Nun muß die neue „po“ Datei noch in eine „mo“ Datei kompiliert werden. Dies wird mit dem Aufruf „./configure && make && cd po && make update-po“ erreicht. Mit „./configure && make distcheck“ kann unser internationalisiertes Sourcecodepaket als „hello-0.1.tar“ ausgeliefert werden.

## 2. Quellen

- Autotools Tutorial --- (sehr hilfreich)

<http://www-src.lip6.fr/homepages/Alexandre.Duret-Lutz/autotools.html>

- A tutorial on Native Language Support using GNU gettext ---

<http://oriya.sarovar.org/docs/gettext/>

- HowTo gettext ---

[http://www.chennaiug.org/wiki/HowTo\\_Gettext](http://www.chennaiug.org/wiki/HowTo_Gettext)