Ocaml-gettext reference manual

Sylvain Le Gall

Ocaml-gettext reference manual

Sylvain Le Gall
Copyright © 2005 Sylvain Le Gall

This user manual is considered as a source code and is licensed under the GNU Lesser General Public License v2.1 with OCaml static compilation exception.

Table of Contents

List of Tables

List of Examples

Chapter 1. Overview

What is ocaml-gettext ?

Ocaml-gettext is a library to support string translation in OCaml. It provides a simple interface to help programmers and translators to create programs that can support different languages. This allows the internationalisation of programs.

OCaml-gettext provides two main functionalities:

- translate English strings into localised strings (depending on which gettext data are installed),
- convert charset from the one used by the translator into the charset of the user.

The library is build according three points of view:

- for the programmer: this library provides functions that can be used in OCaml,
- for the translator: this library provides a standard file format (PO file) to help the translator,
- for the user: this library provides a set of command line options to set the language and the charset.

Ocaml-gettext was initially only a wrapper of gettext. It comes with a patch against the source of xgettext to add support of the OCaml language. As i already used this approach, i was convinced that this library should be better integrated by using more advanced features of OCaml (such as **camlp4**).

As a result of porting gettext to ocaml-gettext, we have :

- a library that can understood the native format of gettext file (MO file),
- two implementations: a wrapper around gettext and a pure OCaml implementation using camomile,
- **ocaml-gettext**: a command line tool to help you to extract, to merge and to install gettext data.

How is ocaml-gettext related to gettext ?

Ocaml-gettext is a close cousin of gettext. In fact, the API is based on gettext. Almost everything in ocaml-gettext is compatible with gettext:

- functions provided in the API are very close to the gettext one,
- the library contains a binding of the gettext library,
- all the file used are gettext compatible: the library uses PO file for translator and MO file for translation,
- the library tries to use the same initialisation sequence as gettext : it uses the same environment variable, tries to find MO files in the same places...

This documentation will not covered the point that are better explained in the gettext documentation. It is highly recommended to read this documentation, before reading this manual. Most of the point that are explained there are not explained again here. However, we have tried to be as precise as possible to enable programming with ocaml-gettext without having the need to be a gettext expert.

Chapter 2. How to install ocaml-gettext

Using source

To compile ocaml-gettext, you need to install the following prerequisites :

- OCaml [http://caml.inria.fr/ocaml/release.en.html] v3.08.3 or later,
- findlib [http://www.ocaml-programming.de/packages/] v1.0.4 or later,
- ocaml-fileutils [http://www.gallu.homelinux.org/download/] v0.3.0 or later,
- ocaml-ast-analyze [http://www.gallu.homelinux.org/download/] v0.1.0 or later,
- camomile [http://prdownloads.sourceforge.net/camomile/] v0.6.0 or later,
- gettext [http://www.gnu.org/software/gettext/gettext.html#TOCdownloading] v0.14.3 or later,
- OUnit [http://www.xs4all.nl/~mmzeeman/ocaml/] v1.0.1 or later [1],
- ocaml-benchmark [http://sourceforge.net/project/showfiles.php?group_id=117069] v0.6 or later [2],
- Docbook DTD and stylesheets [http://prdownloads.sourceforge.net/docbook/] v4.3 [3],
- xsltproc [http://xmlsoft.org/XSLT/downloads.html] v1.1.12 or later, [3],
- fop [http://www.apache.org/dyn/closer.cgi/xml/fop] v0.20.5 or later. [3],

Camomile and gettext are optional but you need at least one of them in order to be able to build the command line tool **ocaml-gettext**.

After having build and install all prerequisites, extract the source code of ocaml-gettext to a directory. Go to the source directory and type :

- **./configure** [4]
- **make**
- **make install**
- **cd test**
- **./test** [1]
- **./benchmark** [2]

Since ocaml-gettext is not yet a stable release, building the benchmark and the unitary test tools is important, especially for debugging purpose. If you encounter problems, you should first try to run this two commands. They will give you good hints on what causes the problem.

[1] Only if you want to build unitary test tool
[2] Only if you want to build benchmarking tool
[3] Only if you want to build the documentation
[4] If needed, you can use **./configure `--help`** to have a complete help on every option you can use to tweak the installation of ocaml-gettext. To enable documentation generation use `--enable-doc`. To enable benchmark executable use `--enable-bench`. To enable test executable, use `--enable-test`.

Note

There is a `patches` directory in the source tree. This directory contains patches against different programs

that should be compatible with ocaml-gettext. Please have a look to the `README.patches` of this directory, to know how to handle patching these programs.

Debian distribution

A debian package is available in the official Debian archive (release "unstable", "main" section).

To install it, use the command : **apt-get `install` `libgettext-ocaml-dev`** .

Other distributions

There is no plan to release packages for other distributions. If you have any skill concerning the packaging of ocaml-gettext for any other distribution, please contact me.

Chapter 3. Programming with ocaml-gettext

Overview

The API of ocaml-gettext is really reduced. It is made on purpose. The design is heavily based on modules and functors. There is no real reason for this design, it was just really useful at the time the code was written.

The library supposes that all the `textdomain` that will be used during translation, are declared before using it. It is a real constraint, but it enables more optimisation. Moreover, it allows having a more "functional" use.

First of all, a parameter `t` should be defined. This parameter holds all the required value to initialise ocaml-gettext. In particular, it contains information about :

- which textdomain will be used,
- which language will be used,
- how the error should be handle,
- which directory to search.

This parameter is build and updated through internal functions. You don't have direct access to it.

The parameter `t` is not directly used for translation. It must be converted into a parameter `t'` which is a real function to access translation. The transformation from `t` to `t'` is handled through a function `realize`. The parameter `t'` is used in low level translation.

All the work of the library is done in the function `realize`. This function is not provided in the base package. It is build out of real implementation of ocaml-gettext (such as `gettext-camomile` or `gettext-stub`). This function could handle all the parameters in different ways. Concerning `gettext-camomile`, it builds a translation table for all the files found which correspond to a declared `textdomain`.

Since it should be very difficult to pass a parameter `t` or `t'` in all functions that should use translation, we provide a more simple way to use the library. The top level functions use a global reference to store the parameters `t` and `t'`. This helps to integrate ocaml-gettext more easily into existing application.

Makefile and source layout

The source layout should conform to the one described in gettext documentation. In particular, it should contain a `po` directory. There should be in this directory :

- a file `LINGUAS` describing available translations,
- a file `POTFILES` containing the listing of all files that contain translatable strings,
- a `Makefile` to build the whole thing,
- a set of PO file which contains translated strings for different languages.

During the build, the `Makefile` should generate a file `your-domain.pot` that contains a template PO file that can be used by translator.

Example 3.1. `Makefile` for `po`

```
##########################################################################
#  Ocaml-gettext : example code                                          #
#                                                                        #
#  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  #
#                                                                        #
#  You can redistribute this library and/or modify it under the terms of #
#  the GNU LGPL v2.1 with the OCaml static compilation exception.        #
#                                                                        #
#  Contact: sylvain@le-gall.net                                          #
##########################################################################

OCAML_GETTEXT_PACKAGE = mydomain
LINGUAS=$(shell cat LINGUAS)
SOURCES=POTFILES

OCAML_GETTEXT=ocaml-gettext
OCAML_GETTEXT_EXTRACT_OPTIONS=
OCAML_GETTEXT_COMPILE_OPTIONS=
OCAML_GETTEXT_INSTALL_OPTIONS=
OCAML_GETTEXT_MERGE_OPTIONS=

BUILDPO=../build/share/locale/

POFILES=$(addsuffix .po,$(LINGUAS))
MOFILES=$(addsuffix .mo,$(LINGUAS))
POTFILE=$(OCAML_GETTEXT_PACKAGE).pot

all: install-buildpo

install: install-po

uninstall: uninstall-po

clean:: clean-po

%.mo: %.po
        $(OCAML_GETTEXT) --action compile $(OCAML_GETTEXT_COMPILE_OPTIONS)    \
        --compile-output $@ $^

%.pot: $(SOURCES)
        $(OCAML_GETTEXT) --action extract $(OCAML_GETTEXT_EXTRACT_OPTIONS)    \
        --extract-pot $@ $^

%.po: $(POTFILE)
        $(OCAML_GETTEXT) --action merge   $(OCAML_GETTEXT_MERGE_OPTIONS)      \
        --merge-pot $(POTFILE) $@

$(BUILDPO):
        mkdir -p $(BUILDPO)

.PRECIOUS: $(POTFILE)

install-buildpo: $(MOFILES) $(BUILDPO)
        $(OCAML_GETTEXT) --action install $(OCAML_GETTEXT_INSTALL_OPTIONS)    \
        --install-textdomain $(OCAML_GETTEXT_PACKAGE)                         \
        --install-destdir $(BUILDPO) $(MOFILES)
```

```
install-po: $(MOFILES)
        $(OCAML_GETTEXT) --action install $(OCAML_GETTEXT_INSTALL_OPTIONS)    \
        --install-textdomain $(OCAML_GETTEXT_PACKAGE)                         \
        --install-destdir $(PODIR) $(MOFILES)

uninstall-po:
        $(OCAML_GETTEXT) --action uninstall $(OCAML_GETTEXT_INSTALL_OPTIONS)  \
        --uninstall-textdomain $(OCAML_GETTEXT_PACKAGE)                       \
        --uninstall-orgdir $(PODIR) $(MOFILES)

clean-po:
        -$(OCAML_GETTEXT) --action uninstall $(OCAML_GETTEXT_INSTALL_OPTIONS) \
        --uninstall-textdomain $(OCAML_GETTEXT_PACKAGE)                       \
        --uninstall-orgdir $(BUILDPO) $(MOFILES)
        -$(RM) $(MOFILES)
```

Example 3.2. `LINGUAS`

```
fr
```

Example 3.3. `POTFILES`

```
../library/library.ml
../gui/gui.ml
../program/program.ml
```

To build programs and libraries with ocaml-gettext, the preferred way is to use ocamlfind. There are five findlib packages:

- gettext.base: the base package of ocaml-gettext. It contains the top level type required to compile any library,
- gettext.extension [1] : a package used to extend ocaml-gettext. It is reserved for very particular functions (such as creating a new `realize` function),
- gettext.extract[1]: a package that enables to create special **camlp4** program (such as **ocaml-xgettext**),
- gettext-stub: an implementation of ocaml-gettext using gettext,
- gettext-camomile: an implementation of ocaml-gettext using camomile. It is a pure ocaml implementation.

[1] This feature is described here for your information only. Since it belongs to low level implementation of ocaml-gettext, it should not be used.

In order to link an application or a library using ocaml-gettext, you should link with one of : gettext.base, gettext-camomile or gettext-stub.

Example 3.4. `Makefile`

```
##########################################################################
#  Ocaml-gettext : example code                                          #
#                                                                        #
#  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  #
#                                                                        #
#  You can redistribute this library and/or modify it under the terms of #
#  the GNU LGPL v2.1 with the OCaml static compilation exception.        #
#                                                                        #
#  Contact: sylvain@le-gall.net                                          #
##########################################################################

all: program

clean:
	$(RM) *.cmi *.cmo program

program: programGettext.ml program.ml
	ocamlfind ocamlc -package "gettext-camomile lablgtk2.init" -linkpkg \
	  -I ../library -I ../gui -o $@ library.cma gui.cma $^
```

Library

Library should use the module `Gettext.Library`. It doesn't need any real implementation of ocaml-gettext. By this way, you can let the library user choose the most appropriate ocaml-gettext implementation. This point is essential : a library could be used as well in a GUI program or in short run command line program. These two examples don't require the same kind of implementation at all: GUI program loads most of their translated strings, command line program only use one among them. So by using the module `Gettext.Library` framework, you don't restrict programs to use one particular implementation of ocaml-gettext.

The library should define, using the functor `Init` provided:

- his textdomain through the `textdomain` value,
- his dependencies through the `dependencies` value,
- if needed his charset and directory binding (but it is not recommended to do so).

After having instantiated the module `Gettext.Library` with the appropriate `Init`, you should use the function provided :

- `s_` : for translating singular strings,
- `f_` : for translating singular strings which will be used with `Printf` function [2],
- `sn_` : for translating plural strings,

[2] Strings which should be used by `Printf` function are checked to be sure that the returned strings are equivalent to the provided English string. In particular, every "%"-symbol should be the same in the provided string and the returned string. If not, it is the untranslated string which is returned.

- `fn_` : for translating plural strings which will be used with `Printf` function[2],

Warning

You must keep the function name `s_`, `f_`, `sn_` and `fn_`. The extraction of translatable strings matches these names. If you don't keep it, the extraction of translatable strings will fail.

Example 3.5. `library.ml`

```
(**************************************************************************)
(*  Ocaml-gettext : example code                                          *)
(*                                                                        *)
(*  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  *)
(*                                                                        *)
(*  You can redistribute this library and/or modify it under the terms of  *)
(*  the GNU LGPL v2.1 with the OCaml static compilation exception.        *)
(*                                                                        *)
(*  Contact: sylvain@le-gall.net                                          *)
(**************************************************************************)

open LibraryGettext.Gettext;;

(* Give access to the init of LibraryGettext *)
let init =
  Gettext.init
;;

(* Example function *)
let library_only_function () =

  (* Two simple examples : singular translation *)
  print_endline (s_ "Hello world !");
  Printf.printf (f_ "Hello %s !\n") "world";

  (* More complicated : plural translation, using strings *)
  print_endline (
     (sn_ "There is " "There are " 2)
    ^(string_of_int 2)
    ^(sn_ "plate." "plates." 2)
  );

  (* More simple forms of plural translation, using printf *)
  Printf.printf (fn_ "There is %d plate.\n" "There are %d plates.\n" 2) 2
;;

(* Another example function : used by program.ml *)
let hello_you name =
  Printf.printf (f_ "Hello %s\n") name
;;
```

Example 3.6. `libraryGettext.ml`

```
(**************************************************************************)
(*  Ocaml-gettext : example code                                          *)
(*                                                                        *)
```

```
(*  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>   *)
(*                                                                         *)
(*  You can redistribute this library and/or modify it under the terms of  *)
(*  the GNU LGPL v2.1 with the OCaml static compilation exception.         *)
(*                                                                         *)
(*  Contact: sylvain@le-gall.net                                           *)
(***************************************************************************)

(* Create the module Gettext, using the textdomain "mydomain" *)
module Gettext = Gettext.Library(struct
  let textdomain   = "mydomain"
  let codeset      = None
  let dir          = None
  let dependencies = Gettext.init
end)
;;
```

All the calls to translation functions, use the textdomain provided in `Init`.

The only constraint when using ocaml-gettext in your library is to provide an access to the value `Gettext.Library.init`. This value is used as dependencies for other libraries and programs that depend on it. For example, since you use the library ocaml-gettext, your primary dependency is the function `init` provided in the top level (the function `Gettext.string_of_exception` is localised).

Warning

If you distribute your library, don't forget to mention that ocaml-gettext will only be able to translate the string defined in your library, if and only if the MO file build with is also installed. If not installed ocaml-gettext is useless.

Program

Program should use ocaml-gettext just as libraries do. The only difference lies in the fact that you should provide a `realize` function in the `InitProgram`. The other difference is that the `init` value is not a dependency that should be used by other program. It is a `Arg` usable value. It allows user to define some important parameters.

Example 3.7. `program.ml`

```
(***************************************************************************)
(*  Ocaml-gettext : example code                                           *)
(*                                                                         *)
(*  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>   *)
(*                                                                         *)
(*  You can redistribute this library and/or modify it under the terms of  *)
(*  the GNU LGPL v2.1 with the OCaml static compilation exception.         *)
(*                                                                         *)
(*  Contact: sylvain@le-gall.net                                           *)
(***************************************************************************)

open ProgramGettext.Gettext;;

let () =
  let my_name = ref ""
  in
  let spf x = Printf.sprintf x
```

```
  in
  let (gettext_args,gettext_copyright) =
    ProgramGettext.Gettext.init
  in
  let args =
    Arg.align (
      [
        "--my-name",
        Arg.String ( fun s ->
          my_name := s
        ),
        ( spf (f_ "name Your name. Default : %S") !my_name )
      ] @ gettext_args
    )
  in
  let () =
    Arg.parse
    args
    ( fun str -> () )
    (
      spf (f_
"\"Hello you\" program by Sylvain Le Gall

%s

Command: program [options]

Options:") gettext_copyright
    )
  in
  Library.hello_you !my_name;
  Gui.hello_you !my_name
;;
```

Example 3.8. `programGettext.ml`

```
(**************************************************************************)
(*  Ocaml-gettext : example code                                          *)
(*                                                                        *)
(*  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  *)
(*                                                                        *)
(*  You can redistribute this library and/or modify it under the terms of  *)
(*  the GNU LGPL v2.1 with the OCaml static compilation exception.        *)
(*                                                                        *)
(*  Contact: sylvain@le-gall.net                                          *)
(**************************************************************************)

(* Create the module Gettext, using the textdomain "mydomain" *)
module Gettext = Gettext.Program
(
  struct
    let textdomain   = "mydomain"
    let codeset      = None
    let dir          = None
    let dependencies = Library.init @ Gui.init
  end
)
(* I do prefer fully ocaml implementation, so choose the
   GettextCamomile module *)
(GettextCamomile.Map)
```

```
;;
```

Example 3.9. Output of program `--help`

```
$>./program --help

    --my-name name                      Your name. Default : ""
    --gettext-failsafe {ignore|inform-stderr|raise-exception}
                                        Choose how to handle failure in
    --gettext-disable                   Disable the translation perform
    --gettext-domain-dir textdomain dir Set a dir to search ocaml-gettex
    --gettext-dir dir                   Add a search dir for ocaml-gette
                                        "/usr/local/share/locale" ].
    --gettext-language language         Set the default language for oca
    --gettext-codeset codeset           Set the default codeset for outp
    -help                               Display this list of options
    --help                              Display this list of options
```

If you want to include a manpage (or info file), that describes the command line option of ocaml-gettext, you should use the Docbook XML fragment distributed with this application. Docbook should be enough generic to allow you to link it into your documentation. If you don't want to link it with your documentation, you can refer to ocaml-gettext-options manpage.

Example 3.10. `program.xml` : Docbook manpage

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE refentry PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"/usr/share/xml/schema/dtd/4.3/docbookx.dtd" [
<!--************************************************************************-->
<!-- Ocaml-gettext : example code                                           -->
<!--                                                                        -->
<!-- Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  -->
<!--                                                                        -->
<!-- You can redistribute this library and/or modify it under the terms of -->
<!-- the GNU LGPL v2.1 with the OCaml static compilation exception.         -->
<!--                                                                        -->
<!-- Contact: sylvain@le-gall.net                                           -->
<!--************************************************************************-->

<!--"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"-->

<!-- Where to find the file ocaml-gettext-options.xml, depends on your installatio
<!ENTITY ocaml_gettext_options_xml "../../doc/ocaml-gettext-options.xml">

]>
<refentry>
  <refmeta>
    <refentrytitle>PROGRAM</refentrytitle>
    <manvolnum>1</manvolnum>
  </refmeta>

  <refnamediv>
```

```
  <refname><command>program</command></refname>
  <refpurpose>programs to say hello.</refpurpose>
</refnamediv>

<refsynopsisdiv>
  <cmdsynopsis>
    <command>program</command>
    <group>
      <arg choice="plain">--my-name <arg choice="req"><replaceable>name</replace
      <arg choice="plain">-help</arg>
      <arg choice="plain">--help</arg>
    </group>
    <sbr/>
    <!-- The ocaml-gettext standard options.-->
    <xi:include
      href="&ocaml_gettext_options_xml;"
      xmlns:xi="http://www.w3.org/2001/XInclude"
      xpointer="xpointer(id('ocaml-gettext-options-cmdsynopsis')/node())"/>
  </cmdsynopsis>
</refsynopsisdiv>

<refsect1>
  <title>DESCRIPTION</title>

  <para>
    This manual page documents briefly the <command>program</command> command.
  </para>

  <variablelist>
    <varlistentry>
      <term>
        <option>--my-name <parameter>name</parameter></option>
      </term>
      <listitem>
        <para>
          Your name. This could be any name, with which you should be greeted.
        </para>
      </listitem>
    </varlistentry>
    <varlistentry>
      <term>
        <option>-help</option>
      </term>
      <term>
        <option>--help</option>
      </term>
      <listitem>
        <para>
          Display the help about the <command>program</command> command.
        </para>
      </listitem>
    </varlistentry>
  </variablelist>
</refsect1>

<refsect1>
  <xi:include
    href="&ocaml_gettext_options_xml;"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    xpointer="xpointer(id('ocaml-gettext-options-description')/node())" />
</refsect1>

<refsect1>
  <title>SEE ALSO</title>
```

```
    <para>
      <citerefentry>
        <refentrytitle>ocaml-gettext</refentrytitle>
        <manvolnum>5</manvolnum>
      </citerefentry>
    </para>
  </refsect1>
</refentry>
```

Example 3.11. `Makefile` : Makefile for docbook manpage

```
##########################################################################
#  Ocaml-gettext : example code                                          #
#                                                                        #
#  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  #
#                                                                        #
#  You can redistribute this library and/or modify it under the terms of #
#  the GNU LGPL v2.1 with the OCaml static compilation exception.        #
#                                                                        #
#  Contact: sylvain@le-gall.net                                          #
##########################################################################

all: program.1

clean:
        $(RM) program.1

program.1: program.xml
        xmllint --xinclude --noent --noout --postvalid $^
        xsltproc --xinclude /usr/share/xml/docbook/stylesheet/nwalsh/manpages/docb
```

You should take care of what implementation of ocaml-gettext you are using. In order to choose the right implementation you should consider your program and every characteristic of it (how many strings does it need to fetch ? Does it use already a C library that link with gettext ?).

Table 3.1. Characteristics of ocaml-gettext implementation.

Implementation	Characteristics	Use
GettextCamomile.Map	• Pure OCaml implementation, • Full load of all MO files before any translation, • Use OCaml standard `Map`.	• Pure OCaml program, • Program that requires to translate a lot of strings, • Threaded program (since it use OCaml Map, it should be thread safe without problem).
GettextCamomile.Hashtbl	• Pure OCaml implementation,	• Pure OCaml program,

Implementation	Characteristics	Use
	• Full load of all MO file before any translation, • Use OCaml standard `Hashtbl`.	• Program that requires to translate a lot of strings, • Should work with threaded program, provided that the `Hashtbl` works in threaded environment.
GettextCamomile.Open	• Pure OCaml implementation, • Load strings from MO if needed, • Use OCaml standard `Hashtbl`, • Use a dichotomic search for the strings, • Compute MO file to open at initialisation, • Open a file when fetching string, • Doesn't memorise already translated strings, • Implementation design copied from gettext.	• Pure OCaml program, • Program that require to translate very few strings, • Should work with threaded program, provided that `open_in` function call work.
GettextStub.Native	• Native gettext library, • Partial load of all MO file before any translation, use `mmap`.	• OCaml program that uses library compiled with gettext, • Should work with threaded program, provided that the `gettext` work in threaded environment.
GettextStub.Preload	• Native gettext library, • Forced load of all MO file before any translation, the preload is realized by trying to load the string "" for all the textdomain defined.	• OCaml program that uses library compiled with gettext, • Program that needs to translate a lot of strings, • Should work with threaded program, provided that the `gettext` work in threaded environment.

Graphical user interface

Graphical user interface works just as a program or a library. The only difference is that the file which

contains the graphical user interface should not be written in OCaml. You have two cases :

- You use glade file, so you should extract the strings from this file using **xgettext**
- You use a hand written interface written in OCaml, the extraction of the strings follow the same way as a library.

You should use the first alternative : it is easier for a translator to extract strings, without having to compile your application (it enables translators that don't know OCaml to help you). In order to do so, you should use the native **xgettext** binary (provided with gettext). It should support the format of the translatable strings found in your GUI file (for example, **xgettext** supports glade file).

But for now you can only use the second alternative, because OCaml is not yet supported in **xgettext**. This should be fixed, once **ocaml-gettext** will be enough stable to become a back-end for **xgettext** [3].

In the two cases, you just have to add your GUI file (in OCaml or native form) to `POTFILES`.

Graphical user interfaces are good candidates for settings a fixed `Init.codeset`. Typically, GTK2 interfaces require to have these parameters set to UTF-8.

Example 3.12. `gui.ml`

```
(**************************************************************************)
(*  Ocaml-gettext : example code                                          *)
(*                                                                        *)
(*  Copyright (C) 2003, 2004, 2005 Sylvain Le Gall <sylvain@le-gall.net>  *)
(*                                                                        *)
(*  You can redistribute this library and/or modify it under the terms of  *)
(*  the GNU LGPL v2.1 with the OCaml static compilation exception.        *)
(*                                                                        *)
(*  Contact: sylvain@le-gall.net                                          *)
(**************************************************************************)

open GuiGettext.Gettext;;

(* Give access to the init of GuiGettext *)
let init =
  Gettext.init
;;

(* Build a simple window that display your name *)
let hello_you name =
  let spf x = Printf.sprintf x
  in
  let window = GWindow.window ~title:(s_ "Hello world !") ~border_width:12 ()
  in
  let label = GMisc.label ~text:(spf (f_ "Hello %s") name) ~packing:window#add ()
  in
  ignore (window#event#connect#delete ~callback:(fun _ -> false));
  ignore (window#connect#destroy ~callback:(fun _ -> GMain.Main.quit ()));
  window#show ();
  GMain.Main.main ()
;;
```

[3] For now, extracting strings from OCaml source file and glade file, requires to patch gettext™. You can find this path in patches. This patch will be sent to upstream author once it will be considered enough stable.

Example 3.13. ./program --gettext-dir ../build/share/locale --gettext-lang C --my-name Sylvain

Example 3.14. ./program --gettext-dir ../build/share/locale --gettext-lang fr --my-name Sylvain

Warning

If you build lablgtk™ application, you must keep in mind that some locale settings are made in the function `GMain.Init`. Those settings could override those done through the command line options. In order to correctly use ocaml-gettext, you must be sure to call `GMain.Init` before using `Arg.parse` with the ocaml-gettext args.

Chapter 4. Translating ocaml-gettext programs and libraries

Ocaml-gettext has been built around gettext. This allows translators to use exactly the same technics as they should use with gettext. All the documentation required for translating can be found in gettext documentation.

It is recommended to use GUI which allows easier translation such as :

- gtranslator [http://gtranslator.sourceforge.net/],
- kbabel [http://i18n.kde.org/tools/kbabel/].

Chapter 5. Using ocaml-gettext programs

Ocaml-gettext program can be used just as any OCaml program. The only difference with standard OCaml program is that they come with a bunch of command line options which are specific to OCaml program. In most cases, you just have to define a well suited `LC_ALL` or `LANG` environment variable. Since ocaml-gettext is compatible with gettext, if your environment variable works with gettext, it should also works with ocaml-gettext.

You can find more details in the gettext documentation or in the ocaml-gettext-options manpage.

Appendix A. Links

- OCaml website [http://caml.inria.fr/]
- Gettext website [http://www.gnu.org/software/gettext/gettext.html]
- Gettext documentation [http://www.gnu.org/software/gettext/manual/gettext.html]
- Camomile website [http://camomile.sourceforge.net/]
- Ocaml-fileutils website [http://www.gallu.homelinux.org/ocaml-fileutils.html]
- OUnit website [http://www.xs4all.nl/~mmzeeman/ocaml/]
- Ocaml-benchmark website [http://ocaml-benchmark.sourceforge.net/]
- Docbook website [http://docbook.sourceforge.net/]

Appendix B. Manpages

Name

ocaml-gettext — program to manage PO and MO files for Ocaml source files.

Synopsis

`ocaml-gettext` [--version | --short-version | -help | --help
| --action [[extract] | compile | install | uninstall | merge]]
[--extract-command {*`cmd`*}] [--extract-default-option {*`options`*}] [--extract-filename-option {*`filename`*} {*`options`*}] [--extract-pot {*`filename`*}]
[--compile-output {*`filename`*}]
[--install-language {*`language`*}] [--install-category {*`category`*}] [--install-textdomain {*`textdomain`*}] [--install-destdir {*`dirname`*}]
[--uninstall-language {*`language`*}] [--uninstall-category {*`category`*}] [--uninstall-textdomain {*`textdomain`*}] [--uninstall-orgdir {*`dirname`*}]
{--merge-pot {*`filename`*}} [--merge-backup-extension {*`extension`*}]
[--gettext-failsafe [{ignore} | {inform-stderr} | {raise-exception}]] [--gettext-disable] [--gettext-domain-dir {*`textdomain`*} {*`dir`*}] [--gettext-dir {*`dir`*}] [--gettext-language {*`language`*}] [--gettext-codeset {*`codeset`*}]
[file...]

DESCRIPTION

This manual page documents briefly the **ocaml-gettext** command.

`--action` *`extract`*
: Files provided are considered to be Ocaml source files and **ocaml-gettext** tries to extract translatable strings of it. The output of the command is a POT file. As a special case, if a file named `POTFILES` is in the list of the file provided, every line of it is considered as a file to be searched.

`--action` *`compile`*
: Files provided are considered to be PO file. These files are compiled in binary MO files,

`--action` *`install`*
: Files provided are considered to be MO files. They are installed in their respective directories considering language, textdomain and category,

`--action` *`uninstall`*
: This is the symmetric command to `install`, but it uninstalls files provided for the considered language, textdomain and category,

`--action` *`merge`*
: Merges a POT file with the provided PO file.

`--extract-command` *`cmd`*
: Command to extract translatable strings from an Ocaml source file. This command should output the same marshalled structure as **ocaml-xgettext**. The best to do is to use the same build version of **ocaml-gettext**.

`--extract-default-option` *`options`*
: Default options used when extracting translatable strings. These options are **camlp4** options and will be passed to **ocaml-xgettext** when processing files that don't already have specific **camlp4** options.

`-extract-filename-option` *`filename options`*
: Specific filename **camlp4** options. It is used when extracting strings from the specified filename. It overrides default **camlp4** options.

`--extract-pot` *`filename`*
: POT file to write when extracting translatable strings.

`--compile-output` *`filename`*
: MO file to write when compiling a PO file. If not provided, the output will be the name of the PO file with ".mo" extension.

`--install-language` *`language`*
: Language to use when installing a MO file.

`--install-category` *`category`*
: Category to use when installing a MO file.

`--install-textdomain` *`textdomain`*
: Textdomain to use when installing a MO file.

`--install-destdir` *`dirname`*
: Base directory to use when installing a MO file.

`--uninstall-language` *`language`*
: Language to use when uninstalling a MO file.

`--uninstall-category` *`category`*
: Category to use when uninstalling a MO file.

`--uninstall-textdomain` *`textdomain`*
: Textdomain to use when uninstalling a MO file.

`--uninstall-orgdir` *`dirname`*
: Base directory used when uninstalling a MO file.

`--merge-pot` *`filename`*
: POT file to use as a master for merging PO file.

`--merge-backup-extension` *`extension`*
: Backup extension to use when moving PO file which have been merged.

`--version`
: Return version information on ocaml-gettext.

`--short-version`
: Returns only the version of ocaml-gettext. The return is made to be easily parseable by **configure** script. The output of this command will always be the shortest version string, made of numeric characters (0-9) and ".". The version strings should be compared considering that a version A is greater than a version B if there is a number between two "." of A that is greater than B the corresponding number, beginning at the right of the string. For example: 0.14 is greater than 0.13.1.

`-help` , `--help`
: Displays the help about the **ocaml-gettext** command.

OCAML-GETTEXT OPTIONS

This section describes briefly the common options provided by programs using ocaml-gettext library.

`--gettext-failsafe ignore`
: Defines the behaviour of ocaml-gettext regarding any error that could be encountered during the processing of string translation. `ignore` is the default behaviour. The string returned is the original string untranslated. This behaviour is consistent and allows to have a usable output, even if it is not perfect.

`--gettext-failsafe inform-stderr`
: Same behaviour as `ignore`, except that a message is printed on stderr,

`raise-exception` — Stops the program by raising an exception when an error is encountered.

`--gettext-disable` — Disables any translation made by ocaml-gettext. All translations return the original string untranslated.

`--gettext-domain-dir` *`textdomain dir`* — Defines a dir to search for a specific domain. This could be useful if MO files are stored in a non standard directory.

`--gettext-dir` *`dir`* — Adds a directory to search for MO files.

`--gettext-language` *`language`* — Sets the language to use in ocaml-gettext library. The language should be POSIX compliant. The language should follow the following convention : lang[_territory][.charset][@modifier]. The lang and territory should be two letters ISO code. Charset should be a valid ISO character set (at least recognised by the underlying charset recoding routine). For example, valid languages are : fr_FR.ISO-8859-1@euro, de_DE.UTF-8.

`--gettext-codeset` *`codeset`* — Sets the codeset for output.

Users should be aware that these command line options, apply only for strings after the initialisation of the library. This means that if the options initially guessed by ocaml-gettext don't match the command line provided, there should be some untranslated string, because these strings are translated before parsing options. This is particularly true for the usage message itself (`--help`) : even if the strings are translated, they are translated before setting the correct option.

Some options (`--gettext-codeset` for example) are overrided internally for particular use. It should be required to always translate strings to UTF-8 in graphical user interface (because GTK2 requires it).

NOTES

Options `--uninstall-language`, `--uninstall-textdomain`, `--install-language` and `--install-textdomain` could be guessed from the filename provided. You must be aware that these options can conflict with the fact that you provide several files to install. For example, if you provide a textdomain, you should either install several MO files which filenames only reflect the language or only one MO file if you also provide a language. For example, you can execute **ocaml-gettext `--install-textdomain` *`mytextdomain fr.po de.po`*** without problem, but you cannot execute **ocaml-gettext `--install-textdomain` *`mytextdomain`* `--install-language` *`fr fr.po de.po`*** . This restriction is due to the fact that you should not over specified file installation.

Rules for guessing the language/textdomain are : language[.textdomain].mo. For a full automated install without giving any hints, through command line options, you should name your file `fr.mytextdomain.mo` or `de.mytextdomain.mo`.

SEE ALSO

ocaml-xgettext(1)

ocaml-gettext(5)

Name

ocaml-xgettext — program to extract translatable strings from Ocaml source file.

Synopsis

`ocaml-xgettext` [camlp4 arguments] [*`filename`*]

DESCRIPTION

This manual page documents briefly the **ocaml-xgettext**

ocaml-xgettext is a **camlp4** top level. This top level is compiled with the printer module `pr_gettext.cmo`. It outputs an Ocaml marshalled data structure that can only be understood by **ocaml-gettext**. The purpose of this program is to be a back end for Ocaml source code string extraction. You should not use it directly.

SEE ALSO

ocaml-gettext(1)

camlp4(1)

Name

ocaml-gettext — common options to manage internationalisation in Ocaml program through ocaml-gettext library.

Synopsis

[--gettext-failsafe [{ignore} | {inform-stderr} | {raise-exception}]] [--gettext-disable] [--gettext-domain-dir {*textdomain*} {*dir*}] [--gettext-dir {*dir*}] [--gettext-language {*language*}] [--gettext-codeset {*codeset*}]

OCAML-GETTEXT OPTIONS

This section describes briefly the common options provided by programs using ocaml-gettext library.

`--gettext-failsafe ignore`	Defines the behaviour of ocaml-gettext regarding any error that could be encountered during the processing of string translation. `ignore` is the default behaviour. The string returned is the original string untranslated. This behaviour is consistent and allows to have a usable output, even if it is not perfect.
`--gettext-failsafe inform-stderr`	Same behaviour as `ignore`, except that a message is printed on stderr,
`--gettext-failsafe raise-exception`	Stops the program by raising an exception when an error is encountered.
`--gettext-disable`	Disables any translation made by ocaml-gettext. All translations return the original string untranslated.
`--gettext-domain-dir` *`textdomain dir`*	Defines a dir to search for a specific domain. This could be useful if MO files are stored in a non standard directory.
`--gettext-dir` *`dir`*	Adds a directory to search for MO files.
`--gettext-language` *`language`*	Sets the language to use in ocaml-gettext library. The language should be POSIX compliant. The language should follow the following convention : lang[_territory][.charset][@modifier]. The lang and territory should be two letters ISO code. Charset should be a valid ISO character set (at least recognised by the underlying charset recoding routine). For example, valid languages are : fr_FR.ISO-8859-1@euro, de_DE.UTF-8.
`--gettext-codeset` *`codeset`*	Sets the codeset for output.

Users should be aware that these command line options, apply only for strings after the initialisation of the library. This means that if the options initially guessed by ocaml-gettext don't match the command line provided, there should be some untranslated string, because these strings are translated before parsing options. This is particularly true for the usage message itself (`--help`) : even if the strings are translated, they are translated before setting the correct option.

Some options (`--gettext-codeset` for example) are overrided internally for particular use. It should be required to always translate strings to UTF-8 in graphical user interface (because GTK2 requires it).