Package 'potools'

October 14, 2022

Type Package

Title Tools for Internationalization and Portability in R Packages

Version 0.2.2

Author Michael Chirico

Depends R (>= 4.0.0)

Imports data.table

Suggests crayon, knitr, rmarkdown, testthat

SystemRequirements gettext

Maintainer Michael Chirico <MichaelChirico4@gmail.com>

Description Translating messages in R packages is managed using the po top-level directory and the 'gettext' program. This package provides some helper functions for building this support in R packages, e.g. common validation & I/O tasks.

License GPL-3

URL https://github.com/MichaelChirico/potools

BugReports https://github.com/MichaelChirico/potools/issues

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2021-07-12 07:00:02 UTC

R topics documented:

`check_cracked_messages` *Check for cracked messages more suitable for templating*

Description

Diagnose the R messages in a package to discover the presence of "cracked" messages better served for translation by templating. See Details.

Usage

```
check_cracked_messages(message_data)
```

Arguments

`message_data`	A `data.table`, or object convertible to one.

Details

Error messages built like `stop("You gave ", n, " arguments, but ", m, " are needed.")` are in general hard for translators – the correct translation may be in a totally different order (e.g., this is often the case for Japanese). It is preferable instead to use `gettextf` to build a templated message like `stop(gettextf("You gave %d arguments but %d are needed.", n, m))`. Translators are then free to rearrange the template to put the numeric pattern where it fits most naturally in the target language.

Value

A `data.table` with columns `call`, `file`, `line_number`, and `replacement` summarizing the results.

Author(s)

Michael Chirico

See Also

`translate_package`, `update_pkg_po`

Examples

```
pkg <- file.path(system.file(package = 'potools'), 'pkg')
# copy to a temporary location to be able to read/write/update below
tmp_pkg <- file.path(tempdir(), "pkg")
dir.create(tmp_pkg)
file.copy(pkg, dirname(tmp_pkg), recursive = TRUE)

# first, extract message data
message_data = get_message_data(tmp_pkg)
```

```
# now, diagnose the messages for any "cracked" ones
check_cracked_messages(message_data)

# cleanup
unlink(tmp_pkg, recursive = TRUE)
rm(pkg, tmp_pkg, message_data)
```

`check_potools_sys_reqs` *Check if the proper system utilities for running package translation are installed*

Description

potools uses the same gettext command line tools that R itself does to run translation. These are required for translation to work properly; this function is mainly for testing use & checks whether the current environment is equipped for translation.

Usage

```
check_potools_sys_reqs()
```

Details

Specifically, potools relies on these command-line utilities

1. `msgmerge`
2. `msgfmt`
3. `msginit`
4. `msgconv`

Value

`TRUE` if the system is ready for translation, otherwise a message suggesting how to proceed.

Author(s)

Michael Chirico

See Also

`update_pkg_po`

check_untranslated_cat *Check for untranslated messages emitted by cat*

Description

Diagnose the R messages in a package to discover the presence of messages emitted by `cat` which haven't been translated (i.e., passed through `gettext`, `gettextf`, or `ngettext`).

Usage

```
check_untranslated_cat(message_data)
```

Arguments

`message_data`	A `data.table`, or object convertible to one.

Details

The function `cat` is commonly used to emit messages to users (e.g., for a `verbose` mode), but it is not equipped for translation. Instead, messages must first be translated and then emitted. Any character literals found in the package's R code used in `cat` but not translated will be flagged by this function.

For flagged calls, a potential replacement is offered, built using `gettext` or `gettextf` (depending on whether one or more `...` arguments are supplied to `cat`). For the `gettextf` case, the suggested template is always `%s` (string) since this works for all inputs; the author should tighten this to the appropriate `sprintf` template marker as appropriate.

NB: not all `cat` calls are included – in particular, no `cat` call specifying a non-default `file` are flagged, nor are any where the supplied `sep` is not a character literal (e.g., `sep=x` instead of `sep=""`)

Value

A `data.table` with columns `call`, `file`, `line_number`, and `replacement` summarizing the results.

Author(s)

Michael Chirico

See Also

`translate_package`, `update_pkg_po`

Examples

```
pkg <- file.path(system.file(package = 'potools'), 'pkg')
# copy to a temporary location to be able to read/write/update below
tmp_pkg <- file.path(tempdir(), "pkg")
dir.create(tmp_pkg)
file.copy(pkg, dirname(tmp_pkg), recursive = TRUE)

# first, extract message data
message_data = get_message_data(tmp_pkg)

# now, diagnose the messages for any untranslated strings shown through cat()
check_untranslated_cat(message_data)

# cleanup
unlink(tmp_pkg, recursive = TRUE)
rm(pkg, tmp_pkg, message_data)
```

---

check_untranslated_src

Check for cracked messages in C/C++ sources

Description

Diagnose the C/C++ messages in a package to discover untranslated messages

Usage

```
check_untranslated_src(message_data)
```

Arguments

message_data	A data.table, or object convertible to one.

Details

This diagnostic looks for literal char arrays passed to messaging functions (as identified by translate_package) which are not marked for translation (by tagging them for translation with _ or N_ macros). These strings cannot be translated until they are so marked.

Value

A data.table with columns call, file, line_number, and replacement summarizing the results. replacement is NA at this time, i.e., no replacement is provided.

Author(s)

Michael Chirico

See Also

translate_package, update_pkg_po

Examples

```
pkg <- file.path(system.file(package = 'potools'), 'pkg')
# copy to a temporary location to be able to read/write/update below
tmp_pkg <- file.path(tempdir(), "pkg")
dir.create(tmp_pkg)
file.copy(pkg, dirname(tmp_pkg), recursive = TRUE)

# first, extract message data
message_data = get_message_data(
  tmp_pkg,
  custom_translation_functions = list(src = "ReverseTemplateMessage:2")
)

# now, diagnose the messages for any untranslated messages in C/C++
check_untranslated_src(message_data)

# cleanup
unlink(tmp_pkg, recursive = TRUE)
rm(pkg, tmp_pkg, message_data)
```

---

get_message_data *Extract user-visible messages from a package*

Description

This function looks in the R and src directories of a package for user-visible messages and compiles them as a data.table to facilitate analyzing this corpus as such.

Usage

```
get_message_data(
  dir=".",
  custom_translation_functions = list(R = NULL, src = NULL),
  verbose=FALSE
)
```

Arguments

dir	Character, default the present directory; a directory in which an R package is stored.
custom_translation_functions	A list with either/both of two components, R and src, together governing how to extract any non-standard strings from the package. See Details in translate_package.
verbose	Logical, default FALSE. Should extra information about progress, etc. be reported?

Value

A `data.table` with the following schema:

1. `message_source`, `character`, either `"R"` or `"src"`, saying whether the string was found in the R or the src folder of the package
2. `type`, `character`, either `"singular"` or `"plural"`; `"plural"` means the string came from `ngettext` and can be pluralized
3. `file`, `character`, the file where the string was found
4. `msgid`, `character`, the string (character literal or `char` array as found in the source); missing for all `type == "plural"` strings
5. `msgid_plural`, `list(character, character)`, the strings (character literals or `char` arrays as found in the source); the first applies in English for `n=1` (see `ngettext`), while the second applies for `n!=1`; missing for all `type == "singular"` strings
6. `call`, `character`, the full call containing the string that was found
7. `line_number`, `integer`, the line in `file` where the string was found
8. `is_repeat`, `logical`, whether the `msgid` is a duplicate within this `message_source`
9. `is_marked_for_translation`, `logical`, whether the string is marked for translation (e.g., in R, all character literals supplied to a `...` argument in `stop` are so marked)
10. `is_templated`, `logical`, whether the string is templatable (e.g., uses `%s` or other formatting markers)

Author(s)

Michael Chirico

See Also

`translate_package`, `write_po_file`

Examples

```
pkg <- system.file('pkg', package = 'potools')

get_message_data(pkg)

# includes strings provided to the custom R wrapper function catf()
get_message_data(pkg, custom_translation_functions = list(R = "catf:fmt|1"))

# includes untranslated strings provided to the custom
#   C/C++ wrapper function ReverseTemplateMessage()
get_message_data(
  pkg,
  custom_translation_functions = list(src = "ReverseTemplateMessage:2")
)

# cleanup
rm(pkg)
```

`translate_package` *Interactively provide translations for a package's messages*

Description

This function handles the "grunt work" of building and updating translation libraries. In addition to providing a friendly interface for supplying translations, some internal logic is built to help make your package more translation-friendly.

To do so, it builds on low-level command line tools from `gettext`. See Details.

Usage

```
translate_package(
  dir='.', languages,
  diagnostics = list(
    check_cracked_messages,
    check_untranslated_cat,
    check_untranslated_src
  ),
  custom_translation_functions = list(R = NULL, src = NULL),
  max_translations = Inf,
  use_base_rules = package %chin% .potools$base_package_names,
  copyright = NULL, bugs = '', verbose=FALSE
)
```

Arguments

`dir`	Character, default the present directory; a directory in which an R package is stored.
`languages`	Character vector; locale codes to which to translate. See Details.
`diagnostics`	A `list` of diagnostic functions to be run on the package's message data. See Details.
`custom_translation_functions`	A `list` with either/both of two components, `R` and `src`, together governing how to extract any non-standard strings from the package. See Details.
`max_translations`	Numeric; used for setting a cap on the number of translations to be done for each language. Defaults to `Inf`, meaning all messages in the package.
`use_base_rules`	Logical; Should internal behavior match base behavior as strictly as possible? `TRUE` if being run on a base package (i.e., `base` or one of the default packages like `utils`, `graphics`, etc.). See Details.
`copyright`	Character; passed on to `write_po_file`.
`bugs`	Character; passed on to `write_po_file`.
`verbose`	Logical, default `FALSE`. Should extra information about progress, etc. be reported?

Details

translate_package goes through roughly three "phases" of translation.

Phase one is setup – dir is checked for existing translations (toggling between "update" and "new" modes), and R files are parsed and combed for user-facing messages.

Phase two is for diagnostics; see the Diagnostics section below. Any diagnostic detecting "unhealthy" messages will result in a yes/no prompt to exit translation to address the issues before continuing.

Phase three is translation. All of the messages found in phase one are iterated over – the user is shown a message in English and prompted for the translation in the target language. This process is repeated for each domain in languages.

An attempt is made to provide hints for some translations that require special care (e.g. that have escape sequences or use templates). For templated messages (e.g., that use %s), the user-provided message must match the templates of the English message. The templates *don't* have to be in the same order – R understands template reordering, e.g. %2$s says "interpret the second input as a string". See sprintf for more details.

After each language is completed, a corresponding '.po' file is written to the package's 'po' directory (which is created if it does not yet exist).

There are some discrepancies in the default behavior of translate_package and the translation workflow used to generate the '.po'/'.pot' files for R itself (mainly, the suite of functions from tools, update_pkg_po, xgettext2pot, xgettext, and xngettext). They should only be superficial (e.g., whitespace or comments), but nevertheless may represent a barrier to smoothly submitting patchings to R Core. To make the process of translating base R and the default packages (tools, utils, stats, etc.) as smooth as possible, set the use_base_rules argument to TRUE and your resulting '.po'/'.pot'/'.mo' file will match base's.

Custom translation functions:

Some package developers may want to write their own messaging interface, or to use wrappers around the base interface (i.e., stop, warning, message, and a few others) which won't be detected by default (e.g. with update_pkg_po).

In such cases, use the custom_translation_functions argument, whose interface is inspired by the --keyword argument to the xgettext command-line tool. This argument consists of a list with two components, R and src (either can be excluded), owing to differences between R and C/C++. Both components, if present, should consist of a character vector.

For R, there are two types of input: one for named arguments, the other for unnamed arguments.

Entries for named arguments will look like "fname:arg|num" (singular string) or "fname:arg1|num1,arg2|num2" (plural string). fname gives the name of the function/call to be extracted from the R source, arg/arg1/arg2 specify the name of the argument to fname from which strings should be extracted, and num/num1/num2 specify the *order* of the named argument within the signature of fname.

Entries for unnamed arguments will look like "fname:...\xarg1,...,xargn", i.e., fname, followed by :, followed by ... (three dots), followed by a backslash (\), followed by a comma-separated list of argument names. All strings within calls to fname *except* those supplied to the arguments named among xarg1, ..., xargn will be extracted.

To clarify, consider the how we would (redundantly) specify custom_translation_functions for some of the default messagers, gettext, gettextf, and ngettext: custom_translation_functions = list(R = c("gettext:...\domain", "gettextf:fmt|1", "ngettext:msg1|2,msg2|3")).

For src, there is only one type of input, which looks like "fname:num", which says to look at the num argument of calls to fname for char arrays.

Note that there is a difference in how translation works for src vs. R – in R, all strings passed to certain functions are considered marked for translations, but in src, all translatable strings must be explicitly marked as such. So for src translations, custom_translation_functions is not used to customize which strings are marked for translation, but rather, to expand the set of calls which are searched for potentially *untranslated* arrays (i.e., arrays passed to the specified calls that are not explicitly marked for translation). These can then be reported in the check_untranslated_src diagnostic, for example.

Diagnostics:

A diagnostic is a function which takes as input a data.table summarizing the translatable strings in a package (e.g. as generated by get_message_data), evaluates whether these messages are "healthy" in some sense, and produces a digest of "unhealthy" strings and (optionally) suggested replacements.

The diagnostic function must have an attribute named diagnostic_tag that describes what the diagnostic does; it is reproduced in the format Found {nrow(result)} {diagnostic_tag}:. For example, check_untranslated_cat has diagnostic_tag = "untranslated messaging calls passed through cat()".

The output diagnostic result has the following schema:

1. call, character, the call identified as problematic
2. file, character, the file where call was found
3. line_number, integer, the line in file where call was found
4. replacement, character, *optional*, a suggested fix to make the call "healthy"

See check_cracked_messages, check_untranslated_cat, and check_untranslated_src for examples of diagnostics.

Domains:

The input to languages conform to the valid languages accepted by gettext. This almost always takes the form of (1) an ISO 639 2-letter language code; or (2) ll_CC, where ll is an ISO 639 2-letter language code and CC is an ISO 3166 2-letter country code e.g. es for Spanish, es_AR for Argentinian Spanish, ro for Romanian, etc. See Sys.getlocale for some helpful tips about how to tell which locales are currently available on your machine, and see the References below for some web resources listing more locales.

Note also the advice given in the R Installation and Administration manual (also cited below) – if you are writing Spanish translations, a typical package should use language = "es" to generate Spanish translations for *all* Spanish domains. If you want to add more regional flair to your messaging, you can do so through supplemental .po files. For example, you can add some Argentinian messages to es_AR; users running R in the es_AR locale will see messages specifically written for es_AR first; absent that, the es message will be shown; and absent that, the default message (i.e., in the language written in the source code, usually English).

Chinese is a slightly different case – typically, the zh_CN domain is used to write with simplified characters while zh_TW is used for traditional characters. In principal you could leverage zh_TW for Taiwanisms and zh_HK for Hongkieisms.

Currently, translation is limited to the same set of domains as is available for base R: Danish, German, English, British English, Spanish, Farsi, French, Italian, Japanese, Korean, Dutch, Polish, Brazilian Portugese, Russian, Turkish, Mainland Chinese, and Taiwanese Chinese.

This list can be expanded; please file an Issue request on GitHub.

Value

This function returns nothing invisibly. As a side effect, a '.pot' file is written to the package's 'po' directory (updated if one does not yet exist, or created from scratch otherwise), and a '.po' file is written in the same directory for each element of `languages`.

Author(s)

Michael Chirico

References

https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Internationalization
https://cran.r-project.org/doc/manuals/r-release/R-admin.html#Internationalization
https://cran.r-project.org/doc/manuals/r-release/R-ints.html#Internationalization-in-the-R-sources
https://developer.r-project.org/Translations30.html
https://www.isi-web.org/publications/glossary-of-statistical-terms
https://www.gnu.org/software/gettext/
https://www.gnu.org/software/gettext/manual/html_node/Usual-Language-Codes.html#Usual-Language-Codes
https://www.gnu.org/software/gettext/manual/html_node/Country-Codes.html#Country-Codes
https://www.stats.ox.ac.uk/pub/Rtools/goodies/gettext-tools.zip
https://saimana.com/list-of-country-locale-code/

See Also

get_message_data, write_po_file, xgettext, update_pkg_po, checkPoFile, gettext

Examples

```
pkg <- system.file('pkg', package = 'potools')
# copy to a temporary location to be able to read/write/update below
tmp_pkg <- file.path(tempdir(), "pkg")
dir.create(tmp_pkg)
file.copy(pkg, dirname(tmp_pkg), recursive = TRUE)

# run translate_package() without any languages
# this will generate a .pot template file and en@quot translations (in UTF-8 locales)
# we can also pass empty 'diagnostics' to skip the diagnostic step
# (skip if gettext isn't available to avoid an error)
if (isTRUE(check_potools_sys_reqs)) {
  translate_package(tmp_pkg, diagnostics = NULL)
}

## Not run:
# launches the interactive translation dialog for translations into Estonian:
```

```
translate_package(tmp_pkg, "et_EE", diagnostics = NULL, verbose = TRUE)

## End(Not run)

# cleanup
unlink(tmp_pkg, recursive = TRUE)
rm(pkg, tmp_pkg)
```

`write_po_file` *Write a .po or .pot file corresponding to a message database*

Description

Serialize a message database in the '`.po`' and '`.pot`' formats recognized by the gettext ecosystem.

Usage

```
write_po_file(
  message_data, po_file, metadata,
  width = 79L, wrap_at_newline = TRUE,
  use_base_rules = metadata$package %chin% .potools$base_package_names
)
po_metadata(
  package='', version='', language='',
  author='', email='',
  bugs='', copyright = NULL,
  ...
)

## S3 method for class 'po_metadata'
format(x, template = FALSE, use_plurals = FALSE, ...)
## S3 method for class 'po_metadata'
print(x, ...)
```

Arguments

`message_data`	`data.table`, as returned from `get_message_data`. NB: R creates separate domains for R and C/C++ code; it is recommended you do the same by filtering the `get_message_data` output for `message_source == "R"` or `message_source == "src"`. Other approaches are untested.
`po_file`	Character vector giving a destination path. Paths ending in '`.pot`' will be written with template files (e.g., `msgstr` entries will be blanked).
`metadata`	A `po_metadata` object as returned by `po_metadata()`.
`width`	Numeric governing the wrapping width of the output file. Default is `79L` to match the behavior of the `xgettext` utility. `Inf` turns off wrapping (except for file source markers comments).

wrap_at_newline
: Logical, default TRUE to match the xgettext utility's behavior. If TRUE, any msgid or msgstr will always be wrapped at an internal newline (i.e., literally matching \n).

use_base_rules
: Logical; Should internal behavior match base behavior as strictly as possible? TRUE if being run on a base package (i.e., base or one of the default packages like utils, graphics, etc.). See Details.

package
: Character; the name of the package being translated.

version
: Character; the version of the package being translated.

language
: Character; the language of the msgstr. See translate_package for details.

author
: Character; an author (combined with email) to whom to attribute the translations (as Last-Translator).

email
: Character; an e-mail address associated with author.

bugs
: Character; a URL where issues with the translations can be reported.

copyright
: An object used to construct the initial Copyright reference in the output. If NULL, no such comment is written. If a list, it should the following structure:

  - year - [Required] A year or hyphen-separated range of years
  - holder - [Required] The name of the copyright holder
  - title - [Optional] A title for the '.po'
  - additional - [Optional] A character vector of additional lines for the copyright comment section

  If a character scalar, it is interpreted as the holder and the year is set as the POT-Creation-Date's year.

...
: Additional (named) components to add to the metadata. For print.po_metadata, passed on to format.po_metadata

x
: A po_metadata object.

template
: Logical; format the metadata as in a '.pot' template?

use_plurals
: Logical; should the Plural-Forms entry be included?

Details

Three components are set automatically if not provided:

- pot_timestamp - A POSIXct used to write the POT-Creation-Date entry. Defaults to the Sys.time at run time.
- po_timestamp - A POSIXct used to write the PO-Revision-Date entry. Defaults to be the same as pot_timestamp.
- language_team - A string used to write the Language-Team entry. Defaults to be the same as language; if provided manually, the format LANGUAGE <LL@li.org> is recommended.

The charset for output is always set to "UTF-8"; this is intentional to make it more cumbersome to create non-UTF-8 files.

Value

For po_metadata, an object of class po_metadata that has a format method used to serialize the metadata.

Author(s)

Michael Chirico

References

https://www.gnu.org/software/gettext/manual/html_node/Header-Entry.html

See Also

translate_package, get_message_data, xgettext2pot, update_pkg_po

Examples

```
message_data <- get_message_data(system.file('pkg', package='potools'))
desc_data <- read.dcf(system.file('pkg', 'DESCRIPTION', package='potools'), c('Package', 'Version'))
metadata <- po_metadata(
  package = desc_data[, "Package"], version = desc_data[, "Version"],
  language = 'ar_SY', author = 'R User', email = 'ruser@gmail.com',
  bugs = 'https://github.com/ruser/potoolsExample/issues'
)

# add fake translations
message_data[type == "singular", msgstr := "<arabic translation>"]
# Arabic has 6 plural forms
message_data[type == "plural", msgstr_plural := .(as.list(sprintf("<%d translation>", 0:5)))]

# Preview metadata
print(metadata)
# write .po file
write_po_file(
  message_data[message_source == "R"],
  tmp_po <- tempfile(fileext = '.po'),
  metadata
)
writeLines(readLines(tmp_po))

# write .pot template
write_po_file(
  message_data[message_source == "R"],
  tmp_pot <- tempfile(fileext = '.pot'),
  metadata
)
writeLines(readLines(tmp_pot))

# cleanup
file.remove(tmp_po, tmp_pot)
```

```
    rm(message_data, desc_data, metadata, tmp_po, tmp_pot)
```

Index