Package 'potools'

July 23, 2025

Type Package

Title Tools for Internationalization and Portability in R Packages

Version 0.2.4

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,
https://michaelchirico.github.io/potools/

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

Depends R (>= 4.0.0)

Imports data.table, glue

Suggests crayon, knitr, rmarkdown, testthat (>= 3.1.5), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

SystemRequirements gettext

NeedsCompilation no

Author Michael Chirico [cre, aut],
Hadley Wickham [aut]

Maintainer Michael Chirico <MichaelChirico4@gmail.com>

Repository CRAN

Date/Publication 2023-10-30 08:10:10 UTC

Contents

`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 `base::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(which = SYSTEM_REQUIREMENTS)
```

Arguments

`which`	Which requirements to test for. Defaults to all of the command-line utilities on which potools relies, namely, • `msgmerge` • `msgfmt` • `msginit` • `msgconv`

Details

Specifically, potools relies on these command-line utilities:

Value

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

Author(s)

Michael Chirico

See Also

`tools::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, for example if the author knows the input is an integer, use `%d` or `%i` instead of `%s`.

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::data.table()` to facilitate analyzing this corpus as such.

Usage

```
get_message_data(
  dir = ".",
  custom_translation_functions = list(R = NULL, src = NULL),
  style = NULL,
  verbose = !is_testing()
)
```

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()`.
`style`	Translation style, either `"base"` or `"explict"`. The default, `NULL`, reads from the `DESCRIPTION` field `Config/potools/style` so you can specify the style once for your package. Both styles extract strings explicitly flagged for translation with `gettext()` or `ngettext()`. The base style additionally extracts strings in calls to `stop()`, `warning()`, and `message()`, and to `stopf()`, `warningf()`, and `messagef()` if you have added those helpers to your package. The explicit style also accepts `tr_()` as a short hand for `gettext()`. See `vignette("developer")` for more details.
`verbose`	Logical, default `TRUE` (except during testing). Should extra information about progress, etc. be reported?

Value

A `data.table` with the following schema:

- `message_source`: `character`, either `"R"` or `"src"`, saying whether the string was found in the R or the src folder of the package
- `type`: `character`, either `"singular"` or `"plural"`; `"plural"` means the string came from `ngettext()` and can be pluralized
- `file`: `character`, the file where the string was found
- `msgid`: `character`, the string (character literal or `char` array as found in the source); missing for all `type == "plural"` strings
- `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
- `call`: `character`, the full call containing the string that was found
- `line_number`: `integer`, the line in `file` where the string was found
- `is_repeat`: `logical`, whether the `msgid` is a duplicate within this `message_source`
- `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)
- `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)
```

---

po_compile *Compile `.po` files to `.mo`*

Description

This function compiles the plain text `.po` files that translators work with into the binary `.mo` files that are installed with packages and used for live translations.

Usage

```
po_compile(dir = ".", package = NULL, lazy = TRUE, verbose = TRUE)
```

Arguments

`dir`	Path to package root directory.
`package`	Name of package. If not supplied, read from `DESCRIPTION`.
`lazy`	If `TRUE`, only `.mo` functions that are older than `.po` files be updated
`verbose`	If `TRUE`, print information as it goes.

po_create *Create a new .po file*

Description

po_create() creates a new po/{languages}.po containing the messages to be translated.

Generally, we expect you to use po_create() to create new .po files but if you call it with an existing translation, it will update it with any changes from the .pot. See po_update() for details.

Usage

```
po_create(languages, dir = ".", verbose = !is_testing())
```

Arguments

languages	Language identifiers. These are typically two letters (e.g. "en" = English, "fr" = French, "es" = Spanish, "zh" = Chinese), but can include an additional suffix for languages that have regional variations (e.g. "fr_CN" = French Canadian, "zh_CN" = simplified characters as used in mainland China, "zh_TW" = traditional characters as used in Taiwan.)
dir	Character, default the present directory; a directory in which an R package is stored.
verbose	Logical, default TRUE (except during testing). Should extra information about progress, etc. be reported?

po_explain_plurals *Explain plural message criteria verbally*

Description

The nplural syntax in .po file metadata can be hard to grok, even for native speakers. This function tries to de-mystify this by providing verbal expressions of which numbers apply to which index in the msgstr array.

Usage

```
po_explain_plurals(language, index)
```

Arguments

language	A single locale code. See translate_package() for details.
index	Optional. If supplied, a 0-based index to explain for a given language. If not supplied, all plurals for the supplied language are described.

po_extract *Extract messages for translation into a .pot file*

Description

po_extract() scans your package for strings to be translated and saves them into a .pot template file (in the package's po directory). You should never modify this file by hand; instead modify the underlying source code and re-run po_extract().

If you have existing translations, call po_update() after po_extract() to update them with the changes.

Usage

```
po_extract(
  dir = ".",
  custom_translation_functions = list(),
  verbose = !is_testing(),
  style = NULL
)
```

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 TRUE (except during testing). Should extra information about progress, etc. be reported?
style	Translation style, either "base" or "explict". The default, NULL, reads from the DESCRIPTION field Config/potools/style so you can specify the style once for your package. Both styles extract strings explicitly flagged for translation with gettext() or ngettext(). The base style additionally extracts strings in calls to stop(), warning(), and message(), and to stopf(), warningf(), and messagef() if you have added those helpers to your package. The explicit style also accepts tr_() as a short hand for gettext(). See vignette("developer") for more details.

Value

The extracted messages as computed by get_message_data(), invisibly.

po_update — *Update all .po files with changes in .pot*

Description

`po_update()` updates existing `.po` file after the `.pot` file has changed. There are four cases:

- New messages: added with blank `msgstr`.
- Deleted messages: marked as deprecated and moved to the bottom of the file.
- Major changes to existing messages: appear as an addition and a deletion.
- Minor changes to existing messages: will be flagged as fuzzy.

  ```
  #, fuzzy, c-format
  #| msgid "Generating en@quot translations"
  msgid "Updating '%s' %s translation"
  msgstr "memperbarui terjemahan bahasa en@quot..."
  ```

  The previous message is given in comments starting with `#|`. Translators need to update the actual (uncommented) `msgstr` manually, using the old `msgid` as a potential reference, then delete the old translation and the `fuzzy` comment (c-format should remain, if present).

Usage

```
po_update(dir = ".", lazy = TRUE, verbose = !is_testing())
```

Arguments

`dir`	Character, default the present directory; a directory in which an R package is stored.
`lazy`	If `TRUE`, only `.po` files that are older than their corresponding `.pot` file will be updated.
`verbose`	Logical, default `TRUE` (except during testing). Should extra information about progress, etc. be reported?

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 get started, the package developer should run `translate_package()` on your package's source to produce a template `.pot` file (or files, if your package has both R and C/C++ messages to translated), e.g.

To add translations in your desired language, include the target language: in the `translate_package(languages = "es")` call.

Usage

```
translate_package(
  dir = ".",
  languages = NULL,
  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 = !is_testing()
)
```

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. Must be a valid language 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 `base::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.
`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 `TRUE` (except during testing). Should extra information about progress, etc. be reported?

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`.

Phases

translate_package() goes through roughly three "phases" of translation.

1. Setup – dir is checked for existing translations (toggling between "update" and "new" modes), and R files are parsed and combed for user-facing messages.
2. 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.
3. 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, tools::update_pkg_po(), tools::xgettext2pot(), tools::xgettext(), and tools::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

base R provides several functions for messaging that are natively equipped for translation (they all have a domain argument): stop(), warning(), message(), gettext(), gettextf(), ngettext(), and packageStartupMessage().

While handy, some developers may prefer to write their own functions, or to write wrappers of the provided functions that provide some enhanced functionality (e.g., templating or automatic wrapping). In this case, the default R tooling for translation (xgettext(), xngettext() xgettext2pot(), update_pkg_po() from tools) will not work, but translate_package() and its workhorse get_message_data() provide an interface to continue building translations for your workflow.

Suppose you wrote a function stopf() that is a wrapper of stop(gettextf()) used to build templated error messages in R, which makes translation easier for translators (see below), e.g.:

```
stopf = function(fmt, ..., domain = NULL) {
  stop(gettextf(fmt, ...), domain = domain, call. = FALSE)
}
```

Note that potools itself uses just such a wrapper internally to build error messages! To extract strings from calls in your package to stopf() and mark them for translation, use the argument custom_translation_functions:

```
get_message_data(
  '/path/to/my_package',
  custom_translation_functions = list(R = 'stopf:fmt|1')
)
```

This invocation tells `get_message_data()` to look for strings in the `fmt` argument in calls to `stopf()`. `1` indicates that `fmt` is the first argument.

This 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

Cracked messages:

A cracked message is one like:

```
stop("There are ", n, " good things and ", m, " bad things.")
```

In its current state, translators will be asked to translate three messages independently:

- "There are"
- "good things and"
- "bad things."

The message has been cracked; it might not be possible to translate a string as generic as "There are" into many languages – context is key!

To keep the context, the error message should instead be build with `gettextf` like so:

```
stop(domain=NA, gettextf("There are %d good things and %d bad things."))
```

Now there is only one string to translate! Note that this also allows the translator to change the word order as they see fit – for example, in Japanese, the grammatical order usually puts the verb last (where in English it usually comes right after the subject).

`translate_package` detects such cracked messages and suggests a `gettextf`-based approach to fix them.

Untranslated R messages produced by `cat()`:

Only strings which are passed to certain `base` functions are eligible for translation, namely `stop`, `warning`, `message`, `packageStartupMessage`, `gettext`, `gettextf`, and `ngettext` (all of which have a `domain` argument that is key for translation).

However, it is common to also produce some user-facing messages using `cat` – if your package does so, it must first use `gettext` or `gettextf` to translate the message before sending it to the user with `cat`.

`translate_package` detects strings produced with `cat` and suggests a `gettext`- or `gettextf`-based fix.

Untranslated C/C++ messages:

This diagnostic detects any literal `char` arrays provided to common messaging functions in C/C++, namely `ngettext()`, `Rprintf()`, `REprintf()`, `Rvprintf()`, `REvprintf()`, `R_ShowMessage()`, `R_Suicide()`, `warning()`, `Rf_warning()`, `error()`, `Rf_error()`, `dgettext()`, and `snprintf()`. To actually translate these strings, pass them through the translation macro `_`.

NB: Translation in C/C++ requires some additional `#include`s and declarations, including defining the `_` macro. See the Internationalization section of Writing R Extensions for details.

Custom 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:

- `call`: `character`, the call identified as problematic
- `file`: `character`, the file where `call` was found
- `line_number`: `integer`, the line in `file` where `call` was found
- `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.

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://web.archive.org/web/20230108213934/https://www.isi-web.org/resources/glossary-of-statistica
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(), tools::xgettext(), tools::update_pkg_po(), tools::checkPoFile(), base::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()`, `tools::xgettext2pot()`, `tools::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