String Constants: GUI Internationalization

Version 9.4.0.5

November 10, 2026

This library provides the facility for multiple languages in DrRacket's GUI.

1 Using String Constants

```
(require string-constants)
```
package: string-constants-lib

```
(string-constant name)
```

This form returns the string constant named *name* for `this-language`.

```
(string-constants name)
```

This form returns a list of string constants, one for each language that DrRacket's GUI supports.

```
(string-constant-in-current-language? name)
```

Produces `#t` if `key` has been translated for `this-language`.

```
(dynamic-string-constant name) → string?
  name : string-constant?
```

This, like `string-constant`, returns the string constant named *name*, but without any compile-time checking on the argument.

```
(dynamic-string-constants name) → (listof string?)
  name : string-constant?
```

This, like `string-constants`, returns the string constants matching *name*, but without any compile-time checking on the argument.

```
(string-constant? v) → boolean?
  v : any/c
```

Returns `#t` if *v* is a symbol naming a known string constant.

```
(dynamic-string-constant-in-current-language? key) → boolean?
  key : string-constant?
```

Like `string-constant-in-current-language?`, but without compile-time checking.

```
(string-constant-language? v) → boolean?
  v : any/c
```

Determines if *v* is a symbol naming a language that is supported by the string constants library. The symbol `'english` is guaranteed to return `#t`.

```
(call-with-current-language sc-language
                            thunk)       → any
  sc-language : string-constant-langauge?
  thunk : (-> any)
```

Calls *thunk*; in the dynamic extent of the call, all uses of `string-constant` and `dynamic-string-constant` will use string constants from *sc-language*.

This function is designed to facilitate testing of libraries that use string constants, so they can work regardless of the user's language setting.

```
(this-language)
```

This form returns the name of the current language as a symbol.

```
(all-languages)
```

This form returns a list of symbols (in the same order as those returned from `string-constants`) naming each language.

```
(set-language-pref lang) → void?
  lang : string?
```

Sets the language for the next run of DrRacket to *lang*, which must be a symbol returned from `all-languages`. Does not affect the running DrRacket.

2 Adding String Constants

To add string constants to DrRacket, see the file "private/english-string-constants.rkt" and the other string constants files in the "private" directory. (Some string constants files that have a less permissive license are also available on the pkg server in the string-constants-lib-lgpl pkg.)

Each file has the same format. They are each modules in the string-constants/private/string-constant-lang language. The body of each module is a finite mapping table that gives the mapping from the symbolic name of a string constant to its translation in the appropriate language. Multiple string constants that appear together are implicitly concatenated.

The "english-string-constants" is considered the master file; string constants will be set there and translated into each of the other language files. In addition, the "english-string-constants.rkt" file should contain hints about the context of the strings whose symbol name might not be clear.

To add a new set of string constants, pull requests on the string constants repository are welcome, but adding a definition of string-constants-info to a value that satisfies the contract

```
(listof (list/c symbol? regexp? regexp?
                (and/c module-path? (not/c path-string?))))
```

to a collection-level "info.rkt" file allows string constants to be added via other packages. The first element of each list must be the name of the language, the second and third are regular expressions that are used to match the result of system-language+country to determine if the string constant set should be used (by default). The final element is an absolute module path to the file containing the string constants.

3 Language Environment Variables

- PLTSTRINGCONSTANTS
- STRINGCONSTANTS

If either of these environment variables are set, DrRacket shows you, during startup, which string constants are not yet defined for each language.

You can also specify which languages you are interested in. If either environment variable is bound to a symbol (as interpreted by `read`) you see only the corresponding language's messages. If either one is bound to a list of symbols (again, as interpreted by `read`) you see the messages for all the languages in the list. If either is bound to anything else, you see all of the languages.

The PLTSTRINGCONSTANTS environment variable takes precedence over the STRINGCONSTANTS environment variable.

The PLTSTRINGCONSTANTSLANG controls the language choice, overriding the default saved in the preferences file. If it is not set to one of the languages in the result of `all-languages`, it is ignored.

4 How To Translate DrRacket's String Constants in Your Language

This is a short hands-on guide how to translate the strings used within DrRacket's GUI. It is targeted at people willing to translate the interface who may not know all ins and outs of Racket.

4.1 Introduction

DrRacket is a is a development environment for the many languages you can use via Racket — the language developers' language. The interface is already translated in more than a dozen languages, some of which are more actively maintained than others.

If you are interested how much time and effort you will need to invest — roughly 3 afternoons.

Here are the steps:

4.2 Preparation

Clone the `https://github.com/racket/string-constants` repo. This is where you will work, translate, and finally submit the translation via a pull request.

Go to the `string-constants-lib/string-constants/private` directory — it contains all the translations. Start with copying the file `english-string-constants.rkt` to `yourlanguage-string-constants.rkt`.

English is the master file containing the originals of the strings. Therefore, start with a copy of that. Note that if you continue to update and maintain your translation English will be the file you will sync with.

Once you have copied the file you will need to declare it so DrRacket knows it exists and has to be included with the rest of the translations in the official builds. To do that just edit the file `string-constants/string-constants-lib/string-constants/string-constant.rkt` and add your language in two places:

```
(require (prefix-in english: "private/english-string-
constants.rkt")
...
          (prefix-in yourlanguage: "private/yourlanguage-string-
constants.rkt"))
```

and

```
(define built-in-string-constant-sets
  (list
   (make-sc 'english english:string-constants #f)
...
   (make-sc 'yourlanguage yourlanguage:string-constants #f)))
```

And with that you are done with the initial work — you have created a template for your language (still filled in with the English originals) and you have told Racket to use it. We are off to the races and all we need to do is fill in the translations.

4.3 Translation

Open `yourlanguage-string-constants.rkt` in your favorite text editor. It has a fairly trivial structure: it is basically a map — a long list of key-value pairs. The key is the identifier of the string — how Racket source code is going to refer to the message. Do not change these. The value is the translation. As you started with the English version — you have all the comments and the formatting. I highly recommend keeping these in place because they are very helpful when you need to update the translations.

The first thing you need to do is fix the line:

```
(module english-string-constants "string-constant-lang.rkt"
...)
```

It has to become

```
(module yourlanguage-string-constants "string-constant-lang.rkt"
...)
```

The second thing is to add some metadata to your file. You can use standard comments in the file – either prefix a line with double semicolon `;;` or comment out a block of text via `#| commented block |#`.

In that part of the file put some basic data — like your name, which version of the `private/english-string-constants.rkt` file you are syncing with. This is also the place to put out conventions for the translation such as specific terms, style etc. I would also recommend spelling out the license you want to use. Note that for anything that is to be distributed with Racket, the maintainers will be more likely to accept it if you use the strategy below:

In the Bulgarian translation I have explicitly pointed out:

```
;; This file is distributed under the same terms as Racket
```

Currently translations are double licensed — under the Apache 2.0 license and the MIT license.

All the translated strings are quoted by double quotes. So if you need to put " in the translation you would need to quote it like \". However I recommend you use proper quotation marks for your language.

A new line is represented by \n. ~a is substituted with whatever parameter is given to the string template.

You can split strings — a sequence of strings gets concatenated in a single string so:

```
(install-plt-error-downloading "There was an error when download-
ing the"
                    " .plt file.\n\nDetails:\n")
```

is the same as:

```
(install-plt-error-downloading "There was an error when download-
ing the .plt file.\n\nDetails:\n")
```

You can use this to wrap where you want in case strings get too long. Just make sure you do not have a dangling space at the end of a string portion. So using the current example:

```
(install-plt-error-downloading "There was an error when download-
ing the "
                    " plt file.\n\nDetails:\n")
```

Note how the space is at the end of the first line. This will **cause breakage**. There are ways around this but it is better to not need them.

Once you finish the translation – create a pull request to the https://github.com/racket/string-constants repo. If you are interested in how the whole infrastructure of translation works in DrRacket – have a look at the documentation.