ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

EiffelRSS

PROPERTIES: Developer Guide

Michael Käser <kaeserm@student.ethz.ch>
Martin Luder <luderm@student.ethz.ch>
Thomas Weibel <weibelt@student.ethz.ch>

inf | Informatik
Computer Science

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

PROPERTIES represents a persistent set of properties. The properties can be saved to a file or loaded from a file.

Each key and its corresponding value in the property list is a string.

A property list can contain another property list as its default. This default property list is searched if the property key is not found in the original property list.

PROPERTIES is similar to the `java.util.Properties` class. The main difference is, that PROPERTIES doesn't support keys and values separated by whitespace. It always expects `:` or `=` as a separator between key and value.

See figure 1.1 for an overview of the class

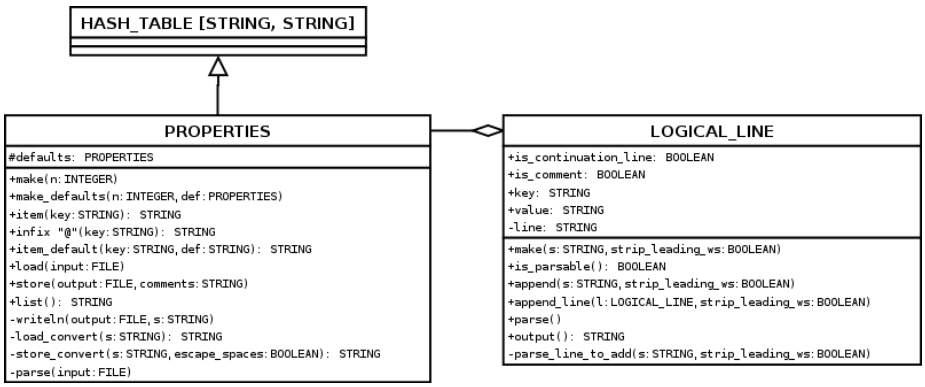

Figure 1.1: UML diagramm of PROPERTIES

1.2 Usage

See the class PROPERTIES_TESTER for a more detailed example.

```
class USAGE_EXAMPLE

create
  make

feature -- Initialization

  make is
      -- Creation procedure.
    do
      -- Open files
      create output_file.make_create_read_write ("↘
      →settings.txt")

      -- Create properties
      create settings.make (10)
      create settings_new.make (10)

      -- Create, display, store and load settings
      -- Put some values
      settings.put ("Benjy, Frankie", "mice")
      settings.put ("Slartibartfast", "fjord designer")
      -- Display properties
      io.put_string (settings.list)
      -- Store properties
      io.put_string ("%NSaving properties%N")
      settings.store (output_file, "PROPERTIES test")
      -- Load properties
      io.put_string ("Loading properties%N%N")
      settings_new.load (output_file)
      -- Display properties
      io.put_string (settings_new.list)
    end

feature -- Arguments

  settings, settings_new: PROPERTIES
      -- Properties

  output_file: PLAIN_TEXT_FILE
      -- Files for input, output and the defaults

end -- class USAGE_EXAMPLE
```

Chapter 2

Features

Because PROPERTIES inherits from HASH_TABLE [STRING, STRING], all features of HASH_TABLE can be applied to a PROPERTIES object, e.g. put, has, replace etc.

2.1 Initialization

2.1.1 make

```
make (n: INTEGER)
  -- Create an empty property table of initial size 'n'
```

2.1.2 make_defaults

```
make_defaults (n: INTEGER; def: PROPERTIES)
  -- Create a property table with 'd' as default values ↘
  →and initial size 'n'
```

2.2 Access

2.2.1 get

```
get (key: STRING): STRING
  -- Item associated with 'key', if present
  -- otherwise default value from 'defaults'
```

Searches for the property with the specified key in this property list. If the key is not found, the default property list, and its defaults, recursively, are checked. It returns Void if the property is not found.

2.2.2 infix "&"

```
infix "&" (key: STRING): STRING
  -- Item associated with 'key', if present
  -- otherwise default value from 'defaults'
```

Same as `get`, but used for infix notation.

2.2.3 get_default

```
get_default (key: STRING; def: STRING): STRING
  -- Item associated with 'key', if present
  -- otherwise default value 'default'
```

Searches for the property with the specified key in this property list. If the key is not found, the default property list, and its defaults, recursively, are checked. It returns `def` if the property is not found.

2.3 Persistence

2.3.1 load

```
load (input: FILE)
  -- Reads a property list from the file 'input'
```

Reads a property list (key and value pairs) from `input`.

`load` processes input in terms of lines. A natural line of input is terminated either by a set of line terminator characters or by the end of the file. A natural line may be either a blank line, a comment line, or hold some part of a key-value pair. The logical line holding all the data for a key-value pair may be spread out across several adjacent natural lines by escaping the line terminator sequence with a backslash character, \. Note that a comment line cannot be extended in this manner. Every natural line that is a comment must have its own comment indicator, as described below. If a logical line is continued over several natural lines, the continuation lines receive further processing, also described below. Lines are read from `input` until end of file is reached.

A natural line that contains only white space characters is considered blank and is ignored. A comment line has an ASCII `#` or `!` as its first non-white space character. Comment lines are also ignored and do not encode key-value information. In addition to line terminators, this feature considers the characters space, tab, and form feed to be white space.

If a logical line is spread across several natural lines, the backslash escaping the line terminator sequence, the line terminator sequence, and any white space at the start of the following line have no affect on the key or value.

The key contains all of the characters in the line starting with the first non-white character and up to, but not including, the first unescaped `=` or `:`. Both key termination characters may be included in the key by escaping them with a preceding backslash character.

For example,

```
\:\=
```

would be the two-character key `":="`. Line terminator characters can be included using `\R` and `\N` escape sequences. Any white space after the key is skipped. If the first non-white space character after the key is `=` or `:`, then it is ignored and any white space after it are also skipped. All remaining characters on the line become part of the associated value string. If there are no remaining characters, the element is the empty string `""`. Once the raw character sequences constituting the key and value are identified, escape processing is performed as described above.

As an example, each of the following three lines specifies the key `"h2g2"` and the associated value `"Douglas Adams"`:

```
h2g2 = Douglas Adams
          h2g2:Douglas Adams
h2g2        :Douglas Adams
```

As another example, the following three lines specify a single property:

```
hitchhikers = Zaphod, \
    Ford, \
    Arthur, \
    Trillian, \
    Marvin
```

The key is `"hitchhikers"` and the associated value is:

```
"Zaphod, Ford, Arthur, Trillian, Marvin"
```

Note that a space appears before each \ so that space will appear after each comma in the final result. The \, line terminator, and leading white space on the continuation line are merely discarded and not replaced by one or more other characters.

As a third example, the line:

```
42
```

specifies the key `"42"` and the associated value `""` (empty string).

2.3.2 store

```
store (output: FILE; comments: STRING)
  -- Writes the property list to the file 'output' and ↘
  →prepends 'comment' and
  -- the actual date as comments
```

Writes the property list to `output` in a format suitable for loading into a property list using the `load` feature. Default properties (`defaults`) are not written out by this feature.

If `comments` is not `Void`, it's prepended by `#` and first written to `output`.

Next, a comment line containing the current date and time is written to `output`.

Then every entry in this `PROPERTIES` table is written out, one per line. For each entry the key string is written, then `=`, then the associated element string. Each character of the key and element string is examined to see whether it should be rendered as an escape sequence. The ASCII character \, tab, form feed, newline, and carriage return are written as \\, \`T`, \`F`, \`N` and \`R`, respectively. For the key, all space characters are written with a preceding \\ character. For the element, leading space characters, but not embedded or trailing space characters, are written with a preceding \ character. The key and element characters `#`, `!`, `=`, and `:` are written with a preceding backslash to ensure that they are properly loaded.

After the entries have been written, `output` is flushed but remains open, after the feature returns.

2.4 Debug

2.4.1 list

```
list: STRING
  -- Returns a string representation of the property ↘
  →list.
```

This feature is especially useful for debugging.

2.5 Arguments

2.5.1 defaults

```
defaults: PROPERTIES
  -- Contains the default values for any keys not found ↘
  →in this property list
```