cp.text

This module provides support for loading, manipulating, and comparing unicode text data.
It works by storing characters with their Unicode 'codepoint `value. In practice, this means that every character is a 64-bit integer, so a` text `value will use substantially more memory than the equivalent encoded` string` value.

The advantages of `text` over `string` representations for Unicode are:

- comparisons, equality checks, etc. actually work for Unicode text and are not encoding-dependent.
- direct access to codepoint values.

The advantages of `string` representations for Unicode are:

- compactness.
- reading/writing to files via the standard `io` library.

Strings and Unicode

LUA has limited built-in support for Unicode text. `string` values are "8-bit clean", which means it is an array of 8-bit characters. This is also how binary data from files is usually loaded, as 8-bit 'bytes'. Unicode characters can be up to 32-bits, so there are several standard ways to represent Unicode characters using 8-bit characters. Without going into detail, the most common encodings are called 'UTF-8' and 'UTF-16'. There are two variations of 'UTF-16', depending on the hardware architecture, known as 'big-endian' and 'little-endian'.

The built-in functions for `string`, such as `match`, `gsub` and even `len` will not work as expected when a string contains Unicode text. As such, this library fills some of the gaps for common operations when working with Unicode text.

Examples

You can convert to and from `string` and `text` values like so:

```lua
local text = require("cp.text")
local simpleString        = "foobar"
local simpleText         = text(stringValue)
local utf8String         = "a丽∀"                -- contains non-ascii characters, defaults to UTF-8.
local unicodeText         = text "a丽∀"             -- contains non-ascii characters, converts from a UTF-8 string.
local utf8String         = tostring(unicodeText) -- `tostring` will default to UTF-8 encoding
local utf16leString         = unicodeText:encode(text.encoding.utf16le) -- or you can be more specific
```

Note that `text` values are not in any specific encoding, since they are stored as 64-bit integer `code-points` rather than 8-bit characers.

Installation

A precompiled version of this module can be found in this directory with a name along the lines of `text-v0.x.tar.gz`. This can be installed by downloading the file and then expanding it as follows:

```
$ cd ~/.hammerspoon # or wherever your Hammerspoon init.lua file is located
$ tar -xzf ~/Downloads/text-v0.x.tar.gz # or wherever your downloads are located
```

If you wish to build this module yourself, and have XCode installed on your Mac, the best way (you are welcome to clone the entire repository if you like, but no promises on the current state of anything else) is to download `init.lua`, `internal.m`, and `Makefile` (at present, nothing else is required) into a directory of your choice and then do the following:

```
$ cd wherever-you-downloaded-the-files
$ [HS_APPLICATION=/Applications] [PREFIX=~/.hammerspoon] make docs install
```

If your Hammerspoon application is located in `/Applications`, you can leave out the `HS_APPLICATION` environment variable, and if your Hammerspoon files are located in their default location, you can leave out the `PREFIX` environment variable. For most people it will be sufficient to just type `make docs install`.

As always, whichever method you chose, if you are updating from an earlier version it is recommended to fully quit and restart Hammerspoon after installing this module to ensure that the latest version of the module is loaded into memory.

Usage

```
text = require("cp.text")
```

Contents

Module Constructors

- text.char(…) -> text
- text.fromCodepoints(codepoints[, i[, j]]) -> text
- text.fromFile(path[, encoding]) -> text
- text.fromString(value[, encoding]) -> text
- text.matcher(pattern[, plain]) -> cp.text.matcher

Module Functions

- text.is(value) -> boolean

Module Methods

- text:encode([encoding]) -> string
- text:find(pattern [, init [, plain]])
- text:len() -> number
- text:match(pattern[, start]) -> …
- text:sub(i [, j]) -> cp.text

Module Constants

- text.encoding

Module Constructors

```
text.char(...) -> text
```

Returns the list of one or more codepoint items into a text value, concatenating the results.

Parameters:

- `...` - The list of codepoint integers.

Returns:

- The `cp.text` value for the list of codepoint values.

```
text.fromCodepoints(codepoints[, i[, j]]) -> text
```

Returns a new `text` instance representing the specified array of codepoints. Since `i` and `j` default to the first
and last indexes of the array, simply passing in the array will convert all codepoints in that array.

Parameters:

- `codepoints` - The array of codepoint integers.
- `i` - The starting index to read from codepoints. Defaults to `1`.
- `j` - The ending index to read from codepoints. Default to `-1`.

Returns:

- A new `text` instance.

Notes:

- You can use a *negative* value for `i` and `j`. If so, it will count back from then end of the `codepoints` array.
- If the codepoint array begins with a Byte-Order Marker (BOM), the BOM is skipped in the resulting text.

```
text.fromFile(path[, encoding]) -> text
```

Returns a new `text` instance representing the text loaded from the specified path. If no encoding is specified,
it will attempt to determine the encoding from a leading Byte-Order Marker (BOM). If none is present, it defaults to UTF-8.

Parameters:

- `value` - The value to turn into a unicode text instance.
- `encoding` - One of the falues from `text.encoding`: `utf8`, `utf16le`, or `utf16be`.

Defaults to `utf8` .

Returns:

- A new `text` instance.

```
text.fromString(value[, encoding]) -> text
```

Returns a new `text` instance representing the string value of the specified value. If no encoding is specified,
it will attempt to determine the encoding from a leading Byte-Order Marker (BOM). If none is present, it defaults to UTF-8.

Parameters:

- `value` - The value to turn into a unicode text instance.
- `encoding` - One of the falues from `text.encoding` : `utf8` , `utf16le` , or `utf16be` . Defaults to `utf8` .

Returns:

- A new `text` instance.

Notes:

- Calling `text(value)` is the same as calling `text.fromString(value, text.encoding.utf8)` , so simple text can be initialized via `local x = text "foo"` when the `.lua` file's encoding is UTF-8.

```
text.matcher(pattern[, plain]) -> cp.text.matcher
```

Returns a matcher for the specified pattern. This follows the conventions of the standard LUA Patterns API. This will return a reusable, compiled parser for the given pattern.

Parameters:

- `pattern` - The pattern to parse
- `plain` - If `true` , the pattern is not parsed and the provided text must match exactly.
  Returns:
    - New `cp.text.matcher` for the pattern.

Module Functions

```
text.is(value) -> boolean
```

Checks if the provided value is a `text` instance.

Parameters:

- `value` - The value to check

Returns:

- `true` if the value is a `text` instance.

Module Methods

```
text:encode([encoding]) -> string
```

Returns the text as an encoded `string` value.

Parameters:

- `encoding` - The encoding to use when converting. Defaults to `cp.text.encoding.utf8` .

```
text:find(pattern [, init [, plain]])
```

Looks for the first match of pattern in the string `value` . If it finds a match, then find returns the indices of `value` where this occurrence starts and ends; otherwise, it returns `nil` . A third, optional numerical argument `init` specifies where to start the search; its default value is `1` and can be negative. A value of `true` as a fourth, optional argument plain turns off the pattern matching facilities, so the function does a plain “find substring” operation, with no characters in pattern being considered “magic”. Note that if plain is given, then `init` must be given as well.

If the pattern has captures, then in a successful match the captured values are also returned, after the two indices.

Preferences:

- `pattern` - The pattern to find.
- `init` - The index to start matching from. Defaults to `1` .

- `plain` - If `true`, the pattern is treated as plain text.

Returns:

- the start index, the end index, followed by any captures

```
text:len() -> number
```

Returns the number of codepoints in the text.

Parameters:

- None

Returns:

- The number of codepoints.

```
text:match(pattern[, start]) -> ...
```

Looks for the first match of the `pattern` in the text value. If it finds one, then match returns the captures from the pattern; otherwise it returns `nil`. If pattern specifies no captures, then the whole match is returned. A third, optional numerical argument `init` specifies where to start the search; its default value is `1` and can be negative.

Parameters:

- `pattern` - The text pattern to process.
- `start` - If specified, indicates the starting position to process from. Defaults to `1`.

Returns:

- The capture results, the whole match, or `nil`.

```
text:sub(i [, j]) -> cp.text
```

Returns the substring of this text that starts at `i` and continues until `j`; `i` and `j` can be negative.
If `j` is absent, then it is assumed to be equal to `-1` (which is the same as the string length).
In particular, the call `cp.text:sub(1,j)` returns a prefix of `s` with length `j`, and `cp.text:sub(-i)` (for a positive `i`) returns a suffix of s with length i.

Module Constants

```
text.encoding
```

The list of supported encoding formats:

- `utf8` - UTF-8. The most common format on the web, backwards compatible with ANSI/ASCII.
- `utf16le` - UTF-16 (little-endian). Commonly used in Windows and Mac text files.
- `utf16be` - UTF-16 (big-endian). Alternate 16-bit format, common on Linux and PowerPC-based architectures.

---

cp.text.matcher

This module provides support for loading, manipulating, and comparing unicode text data.

Installation

A precompiled version of this module can be found in this directory with a name along the lines of `matcher-v0.x.tar.gz`. This can be installed by downloading the file and then expanding it as follows:

```
$ cd ~/.hammerspoon # or wherever your Hammerspoon init.lua file is located
$ tar -xzf ~/Downloads/matcher-v0.x.tar.gz # or wherever your downloads are located
```

If you wish to build this module yourself, and have XCode installed on your Mac, the best way (you are welcome to clone the entire repository if you like, but no promises on the current state of anything else) is to download `init.lua`, `internal.m`, and `Makefile` (at present, nothing else is required) into a directory of your choice and then do the following:

```
$ cd wherever-you-downloaded-the-files
$ [HS_APPLICATION=/Applications] [PREFIX=~/.hammerspoon] make docs install
```

If your Hammerspoon application is located in `/Applications`, you can leave out the

`HS_APPLICATION` environment variable, and if your Hammerspoon files are located in their default location, you can leave out the `PREFIX` environment variable. For most people it will be sufficient to just type `make docs install`.

As always, whichever method you chose, if you are updating from an earlier version it is recommended to fully quit and restart Hammerspoon after installing this module to ensure that the latest version of the module is loaded into memory.

Usage

```
matcher = require("cp.text.matcher")
```

Contents

---

cp.text.matcher

Adapted from ‘utf8.lua’ (https://github.com/Stepets/utf8.lua)

Installation

A precompiled version of this module can be found in this directory with a name along the lines of `matcher-v0.x.tar.gz`. This can be installed by downloading the file and then expanding it as follows:

```
$ cd ~/.hammerspoon # or wherever your Hammerspoon init.lua file is located
$ tar -xzf ~/Downloads/matcher-v0.x.tar.gz # or wherever your downloads are l
ocated
```

If you wish to build this module yourself, and have XCode installed on your Mac, the best way (you are welcome to clone the entire repository if you like, but no promises on the current state of anything else) is to download `init.lua`, `internal.m`, and `Makefile` (at present, nothing else is required) into a directory of your choice and then do the following:

```
$ cd wherever-you-downloaded-the-files
$ [HS_APPLICATION=/Applications] [PREFIX=~/.hammerspoon] make docs install
```

If your Hammerspoon application is located in `/Applications`, you can leave out the `HS_APPLICATION` environment variable, and if your Hammerspoon files are located in their default location, you can leave out the `PREFIX` environment variable. For most people it will be sufficient to just type `make docs install`.

As always, whichever method you chose, if you are updating from an earlier version it is recommended to fully quit and restart Hammerspoon after installing this module to ensure that the latest version of the module is loaded into memory.

Usage

```
matcher = require("cp.text.matcher")
```

Contents

Module Methods

Module Methods

```
matcher:find(value[, start]) -> number, number, ...
```

Processes the text, returning the start position, the end position, followed by any capture group values.

Parameters:

- `value` - The `cp.text` value to process.
- `start` - If specified, indicates the starting position to process from. Defaults to `1`.

Returns:

- The start position for the match, end position, and the list of capture group values.

```
matcher:gmatch(pattern[, start]) -> function
```

Returns an iterator function that, each time it is called, returns the next captures from pattern over string s. If pattern specifies no captures, then the whole match is produced in each call.

Parameters:

- `pattern` - The `cp.text` value to process.

Returns:

- The iterator function.

```
matcher:gmatch(value) -> function
```

Returns an iterator function that, each time it is called, returns the next captures from pattern over string s. If pattern specifies no captures, then the whole match is produced in each call.

Parameters:

- `value` - The `cp.text` value to process.

Returns:

- The iterator function.

```
matcher:gsub(value, repl, limit) -> text, number
```

Returns a copy of `value` in which all (or the first `n` , if given) occurrences of the pattern have been replaced by a replacement string specified by `repl` , which can be text, a string, a table, or a function. gsub also returns, as its second value, the total number of matches that occurred.

Parameters:

- `value` - The text or string value to process.
- `repl` - The replacement text/string/table/function
- `limit` - The maximum number of times to do the replacement. Defaults to unlimited.

Returns:

- `text` - The text value with replacements.
- `number` - The number of matches that occurred.

Notes:

- If repl is text or a string, then its value is used for replacement. The character `%` works as

an escape character: any sequence in repl of the form `%n`, with `n` between `1` and `9`, stands for the value of the `n`-th captured substring (see below). The sequence `%0` stands for the whole match. The sequence `%%` stands for a single `%`.
- If `repl` is a table, then the table is queried for every match, using the first capture as the key; if the pattern specifies no captures, then the whole match is used as the key.
- If `repl` is a function, then this function is called every time a match occurs, with all captured substrings passed as arguments, in order; if the pattern specifies no captures, then the whole match is passed as a sole argument.
- If the value returned by the table query or by the function call is a string or a number, then it is used as the replacement string; otherwise, if it is `false` or `nil`, then there is no replacement (that is, the original match is kept in the string).

```
matcher:match(value[, start]) -> ...
```

Looks for the first match of the pattern in the string `value`. If it finds one, then match returns the captures from the pattern; otherwise it returns `nil`. If pattern specifies no captures, then the whole match is returned. A third, optional numerical argument init specifies where to start the search; its default value is `1` and can be negative.

Parameters:

- `value` - The `cp.text` value to process.
- `start` - If specified, indicates the starting position to process from. Defaults to `1`.

Returns:

- The capture results, the whole match, or `nil`.