cp.utf16

A pure-LUA implementation of UTF-16 decoding

Installation

A precompiled version of this module can be found in this directory with a name along the lines of `utf16-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/utf16-v0.x.tar.gz # or wherever your downloads are loc
ated
```

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

```
utf16 = require("cp.utf16")
```

Contents

Module Functions

Module Functions

```
utf16.char(bigEndian, ...) -> string
```

Receives zero or more integers, converts each one to its corresponding UTF-16 byte sequence and returns a string with the concatenation of all these sequences.

Parameters:

- `bigEndian` - If `true` , the output will list the ‘big’ bytes first
- `...` - The list of UCL codepoint integers to convert.

Returns:

- All the codepoints converted to UTF-16, concatonated into a string.

```
utf16.codepoint(bigEndian, s [, i [, j]]) -> integer...
```

Returns the codepoints (as integers) from all characters in `s` that start between byte position `i` and `j` (both included). The default for `i` is 1 and for `j` is `i` . It raises an error if it meets any invalid byte sequence.

Parameters:

- `bigEndian` - (optional) If set to `true` , the string is encoded in ‘big-endian’ format.
- `s` - The string
- `i` - The starting index. Defaults to `1` .
- `j` - The ending index. Defaults to `i` .

Returns:

- a list of codepoint integers for all characters in the matching range.

```
utf16.codes(bigEndian, s) -> iterator
```

Returns values so that the construction

```
     for p, c in utf16.codes(s) do body end
```

will iterate over all characters in string `s`, with `p` being the position (in bytes) and `c` the code point of each character. It raises an error if it meets any invalid byte sequence.

Parameters:

- `bigEndian` - If `true`, the provided string is in ‘big-endian’ encoding.
- `s` - The string to iterate through.

Returns:

- An iterator

```
utf16.len (bigEndian, s [, i [, j]]) -> number | boolean, number
```

Returns the number of UTF-16 characters in string `s` that start between positions `i` and `j` (both inclusive). The default for `i` is 1 and for `j` is -1. If it finds any invalid byte sequence, returns a false value plus the position of the first invalid byte.

Parameters:

- `bigEndian` - If true, the string is ‘big-endian’.
- `s` - The UTF-16 string
- `i` - The starting index. Defaults to `1`.
- `j` - The ending index. Defaults to `-1`.

Returns:

- the length, or `false` and the first invalid byte index.

```
utf16.offset (bigEndian, s, n [, i]) -> number
```

Returns the position (in bytes) where the encoding of the `n`-th character of `s` (counting from position `i`) starts. A negative `n` gets characters before position `i`. The default for `i` is 1 when `n` is non-negative and `#s + 1` otherwise, so that `utf8.offset(s, -n)` gets the offset of the `n` -th character from the end of the string. If the specified character is neither in the subject nor right after its end, the function returns nil.

As a special case, when `n` is 0 the function returns the start of the encoding of the character that contains the `i` -th byte of `s`.

This function assumes that `s` is a valid UTF-16 string

Parameters:

- `bigEndian` - If `true`, the encoding is 'big-endian'. Defaults to `false`
- `s` - The string
- `n` - The character number to find.
- `i` - The initial position to start from.

Returns:

- The index

---

cp.utf16.be

A pure-LUA implementation of UTF-16 decoding with big-endian ordering.

Installation

A precompiled version of this module can be found in this directory with a name along the lines of `be-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/be-v0.x.tar.gz # or wherever your downloads are locate
d
```

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

```
be = require("cp.utf16.be")
```

Contents

Module Functions

- be.char(...) -> string
- be.codepoint(s [, i [, j]]) -> integer...
- be.codes(s) -> iterator
- be.len (s [, i [, j]]) -> number | boolean, number
- be.offset (s, n [, i]) -> number

Module Functions

```
be.char(...) -> string
```

Receives zero or more integers, converts each one to its corresponding UTF-16 byte sequence and returns a string with the concatenation of all these sequences.

Parameters:

- `...` - The list of UCL codepoint integers to convert.

Returns:

- All the codepoints converted to UTF-16, concatonated into a string.

```
be.codepoint(s [, i [, j]]) -> integer...
```

Returns the codepoints (as integers) from all characters in `s` that start between byte position `i` and `j` (both included). The default for `i` is 1 and for `j` is `i`. It raises an error if it meets any invalid byte sequence.

Parameters:

- `s` - The string
- `i` - The starting index. Defaults to `1`.
- `j` - The ending index. Defaults to `i`.

Returns:

- a list of codepoint integers for all characters in the matching range.

```
be.codes(s) -> iterator
```

Returns values so that the construction

```
    for p, c in utf16.codes(s) do body end
```

will iterate over all characters in string `s`, with `p` being the position (in bytes) and `c` the code point of each character. It raises an error if it meets any invalid byte sequence.

Parameters:

- `s` - The string to iterate through.

Returns:

- An iterator

```
be.len (s [, i [, j]]) -> number | boolean, number
```

Returns the number of UTF-16 characters in string `s` that start between positions `i` and `j` (both inclusive). The default for `i` is 1 and for `j` is -1. If it finds any invalid byte sequence, returns a false value plus the position of the first invalid byte.

Parameters:

- `s` - The UTF-16 string
- `i` - The starting index. Defaults to `1` .
- `j` - The ending index. Defaults to `-1` .

Returns:

- the length, or `false` and the first invalid byte index.

```
be.offset (s, n [, i]) -> number
```

Returns the position (in bytes) where the encoding of the `n` -th character of `s` (counting from position `i`) starts. A negative `n` gets characters before position `i` . The default for `i` is 1 when `n` is non-negative and `#s + 1` otherwise, so that `utf8.offset(s, -n)` gets the offset of the `n` -th character from the end of the string. If the specified character is neither in the subject nor right after its end, the function returns nil.

As a special case, when `n` is 0 the function returns the start of the encoding of the character that contains the `i` -th byte of `s` .

This function assumes that `s` is a valid UTF-16 string

Parameters:

- `s` - The string
- `n` - The character number to find.
- `i` - The initial position to start from.

Returns:

- The index

---

cp.utf16.le

A pure-LUA implementation of UTF-16 decoding with little-endian ordering.

Installation

A precompiled version of this module can be found in this directory with a name along the lines of `le-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/le-v0.x.tar.gz # or wherever your downloads are locate
d
```

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

```
le = require("cp.utf16.le")
```

Contents

Module Functions

- le.char(...) -> string

Module Functions

```
le.char(...) -> string
```

Receives zero or more integers, converts each one to its corresponding UTF-16 byte sequence and returns a string with the concatenation of all these sequences.

Parameters:

- `...` - The list of UCL codepoint integers to convert.

Returns:

- All the codepoints converted to UTF-16, concatonated into a string.

```
le.codepoint(s [, i [, j]]) -> integer...
```

Returns the codepoints (as integers) from all characters in `s` that start between byte position `i` and `j` (both included). The default for `i` is 1 and for `j` is `i`. It raises an error if it meets any invalid byte sequence.

Parameters:

- `s` - The string
- `i` - The starting index. Defaults to `1`.
- `j` - The ending index. Defaults to `i`.

Returns:

- a list of codepoint integers for all characters in the matching range.

```
le.codes(s) -> iterator
```

Returns values so that the construction

```
    for p, c in utf16.codes(s) do body end
```

will iterate over all characters in string `s`, with `p` being the position (in bytes) and `c` the code point of each character. It raises an error if it meets any invalid byte sequence.

Parameters:

- `s` - The string to iterate through.

Returns:

- An iterator

```
le.len (s [, i [, j]]) -> number | boolean, number
```

Returns the number of UTF-16 characters in string `s` that start between positions `i` and `j` (both inclusive). The default for `i` is 1 and for `j` is -1. If it finds any invalid byte sequence, returns a false value plus the position of the first invalid byte.

Parameters:

- `s` - The UTF-16 string
- `i` - The starting index. Defaults to `1`.
- `j` - The ending index. Defaults to `-1`.

Returns:

- the length, or `false` and the first invalid byte index.

```
le.offset (s, n [, i]) -> number
```

Returns the position (in bytes) where the encoding of the `n`-th character of `s` (counting from position `i`) starts. A negative `n` gets characters before position `i`. The default for `i` is 1 when `n` is non-negative and `#s + 1` otherwise, so that `utf8.offset(s, -n)` gets the offset of the `n`-th character from the end of the string. If the specified character is neither in the subject nor right after its end, the function returns nil.

As a special case, when `n` is 0 the function returns the start of the encoding of the character that contains the `i`-th byte of `s`.

This function assumes that `s` is a valid UTF-16 string

Parameters:

- `s` - The string
- `n` - The character number to find.
- `i` - The initial position to start from.

Returns:

- The index