Unihandecode.js Documentation

Release 1.0.1-pre

Jonas Obrist, based on the work by Hiroshi Miura

October 14, 2013

CONTENTS

ASCII transliterations of Unicode text that recognizes CJKV complex characters.

Contents:

CHAPTER

ONE

INTRODUCTION

Unihandecode.js is a port of the unihandecode Python library by Hiroshi Miura and Tomaz Solc, which in turn is a port of the Text::Unidecode Perl module by Sean M. Burke.

The purpose of this library is to transliterate unicode characters to ASCII, with support for complex characters (Japanese, Chinese, Korean and Vietnamese).

1.1 Copyright

The code is licensed under the GPLv3 license, like the original Python library.

1.1.1 Unicode Character Database

Date: 2010-09-23 09:29:58 UDT [JHJ] Unicode version: 6.0.0

Copyright (c) 1991-2010 Unicode, Inc. For terms of use, see http://www.unicode.org/terms_of_use.html For documentation, see http://www.unicode.org/reports/tr44/

1.1.2 Unidecode's character transliteration tables

Copyright 2001, Sean M. Burke <sburke@cpan.org>, all rights reserved.

1.1.3 Python code

Copyright 2010,2011, Hiroshi Miura <miurahr@linux.com> Copyright 2009, Tomaz Solc <tomaz@zemanta.com>

1.1.4 Javascript code

Copyright 2013, Jonas Obrist <ojiidotch@gmail.com>

The programs and documentation in this dist are distributed in the hope that they will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

CHAPTER

TWO

USAGE

2.1 Quickstart

Include dist/unihandecode-1.0.1-pre.core.min.js and at least one decoder source file (dist/unihandecode–1.0.1-pre.<decoder>.min.js, replacing `<decoder>` with the name of your decoder) in your HTML page.

Call `unihandecode.Unihan()` with the name of your decoder as first argument (eg `'ja'` for Japanese), this will return an object which has a `decode` method that takes a string as argument, and returns the transliterated string.

Example:

```
<!DOCTYPE html>
<html>
<body>
<script src="dist/unihandecode-1.0.1-pre.core.min.js"></script>
<script src="dist/unihandecode-1.0.1-pre.ja.min.js"></script>
<script>
    console.log(unihandecode.Unihan('ja').decode('')); // writes 'konnichiha' to the console
    });
</script>
</body>
</html>
```

2.2 List of decoders

- `'ja'`: Japanese Kanji, Hiragana and Katakana support. Supports combined-kanji and full sentences.
- `'zh'`: Chinese Kanji.
- `'kr'`: Korean character support.
- `'vn'`: Vietnamese character support.
- `'diacritic'`: Support for diacritics (eg umlauts).

2.3 The Unihan object

The `unihandecode.Unihan()` function is your main entry point to unihandecode.js. As described above, the first argument it takes is the name of the decoder you wish to use. See *List of decoders* for a list of decoders which are available by default.

It also takes an optional second argument which, if set to `true`, will cause the decoder to throw an error if it failed to decode a character, instead of just skipping that character.

The object returned by `unihandecode.Unihan()` has a single method, `decode`, which takes a string as argument and returns the transliterated string.

CHAPTER

THREE

API

3.1 Introduction

When trying to understand the code, or if you want to write your own decoder, the first thing you should look at is *src/libs/helpers.js* and *src/libs/klass.js*. The first file contains the namespacing mechanism which is used throughout unihandecode.js, and the latter provides the OOP capabilities used in this project.

Note that the Japanese decoder is very different than the other default decoders. So if you want to get started, look at one of the other decoders first.

APIs and files are documented in the order they should be loaded in.

3.2 Libraries

The libraries located in `src/libs` provide helper functionality used throughout the code. Most notably, this includes a namespacing helper and a way to do OOP Javascript code that resembles Python classes. The Python-style OOP libary is used because I'm primarily a Python developer and it makes a lot more sense to me than prototypical inheritance, and because this is a port of a Python library, so keeping the style similar helped a lot.

3.2.1 src/libs/helpers.js

This file **must** be loaded first.

`unihandecode.helpers.`**`module`**`(`*name*, *callback*`)`

Takes a dotted namespace (eg `'unihandecode.mymodule'` and a callback function as arguments. The callback function will be called with a single argument, `scope`, which is an object onto which the module should assign all its public functions, classes and properties. The callback function has no return value.

The same namespace can be defined in multiple files and this function will merge them.

Example:

```
unihandecode.helpers.module('unihandecode.math', function(scope){
    scope.add = function(a, b){
        return a + b;
    };
});
```

`unihandecode.helpers.`**`merge_objects`**`(`*base*, *other*`)`

Merges two objects adding all attributes on `other` onto the `base` object. This function has no return value.

`unihandecode.helpers.`**`startswith`**`(`*string*, *prefix*`)`
Returns `true` if `string` starts with `prefix`, otherwise returns `false`.

`unihandecode.helpers.`**`min`**`(`**args*`)`
Returns the lowest number of the arguments given. All arguments should be integers. Returns `null` if no arguments are given.

`unihandecode.helpers.`**`contains`**`(`*needle*, *haystack*`)`
Returns `true` if `needle` is found in the `haystack` array, otherwise returns `false`.

3.2.2 src/libs/klass.js

This file **must** be loaded second.

Vendored version of https://github.com/ojii/klass. Provides Python style OOP for Javascript.

`Klass``(`**parents*`)`
Returns a class definition function. Takes any number of Klass class constructors as arguments which will be the parent classes of the new class.

The class definition function returned takes an object of properties and methods as arguments and returns a class constructor.

If the class constructor is called, the special `__init__` method is called, so if you want your class to accept arguments and handle them during construction, the object you passed into the class definition function should provide this method.

Every method in a Klass class takes the class instance as explicit first argument, which should be called `self`. This means that `this` resolution is irrelevant to Klass class instances.

Each instance of a Klass class has a sepcial method `$uper` which the name of a method as string as an argument and returns the super method (method from a parent class) for that name.

Example:

```
var BaseClass = Klass()({
    '__init__': function(self, name){
        self.name = name;
    }
});

var MyClass = Klass(BaseClass)({
    'greet': function(self){
        return 'Hi ' + self.name;
    }
});

var myinstance = MyClass('unihandecode.js');
console.log(myinstance.greet()); // writes 'Hi unihandecode.js' to console.
```

`Klass.`**`isinstance`**`(`*instance*, *klass*`)`
Returns `true` if `instance` is an instance of `klass` (or any of the parent classes of `klass`.

`Klass.`**`issubclass`**`(`*subklass*, *klass*`)`
Returns `true` if `subclass` is a subclass of `klass` (or any of parent classes of *klass*‘).

3.3 Base Data

3.3.1 src/base/unicodepoints.js

Contains the basic Unihan code points used by the various decoders.

`unihandecode,base.`**`CODEPOINTS`**

Basic codepoints used by the 'zh', 'vn' and 'kr' decoders.

3.4 Core API

3.4.1 src/basedecoder.js

Defines the base decoder class which is subclassed by all specific decoders.

`unihandecode.`**`BaseDecoder`**`(`*debug*`)`

Base decoder class. When initialized, calls the `unihandecode.BaseDecoder.load_codepoints()` method which is the default entry point for custom decoders.

`unihandecode.BaseDecoder.`**`codepoints`**

Object storing the codepoints used to decode text.

This object maps unicode characters to ASCII text. When a lookup is performed, the character code of the character to look up is right shifted by 8 and transformed to a hexidecimal number. This hex number is then padded with zeroes (on the left) until it is two characters long. This zero-padded hex code is the key used to do the first level lookup in this object. The second level lookup is done by using the character code of the lookup character AND'ed by 255 as an index into an array.

As a result, codepoint objects

`unihandecode.BaseDecoder.`**`load_codepoints`**`()`

Called during initialization and should populate the `unihandecode.BaseDecoder.codepoints` object.

Most decoders only need to override this method.

`unihandecode.BaseDecoder.`**`decode`**`(`*text*`)`

The main API of the decoder. This method should transliterate the `text` given and return the transliterated string.

By default, this method looks for all unicode objects in the string and calls `unihandecode.BaseDecoder.replace_point()` to replace that character.

`unihandecode.BaseDecoder.`**`replace_point`**`(`*character*`)`

Replaces the `character` (a one character string) with an appropriate transliteration, if possible. It does so by looking up the transliteration in `self.codepoints`. The methods `unihandecode.BaseDecoder.code_group()` is used to find the key into `self.codepoints` and `unihandecode.BaseDecoder.grouped_point()` is used to find the index into the array found at that key.

`unihandecode.BaseDecoder.`**`code_group`**`(`*character*`)`

Returns the key into `self.codepoints` for `character`. The key is the hex code of the character code of `character` right shifted by 8 and left-padded with zeros to ensure a key length of two.

`unihandecode.BaseDecoder.`**`grouped_point`**`(`*character*`)`

Returns the index into the array found in `self.codepoints` for `character`. This is done by AND'ing the character code of `character` with 255.

3.4.2 src/unihandecode.js

This file contains the main public API.

`unihandecode.`**`Unihan`**(*lang*, *debug=false*)

Returns a `Unihan` object which can be used to decode. The `lang` argument must be specified and is the name of the decoder to be used. The decoder must be previously registered.

The optional `debug` flag can be set to true, to make the decoding fail in case the decoder can't handle a specific character. The default behavior in that case is to just ignore that character, but with `debug` enabled an error will be thrown.

`unihandecode.Unihan.`**`decode`**(*text*)

Decodes the given string and returns the transliterated version. If `debug` is enabled, this may throw errors.

`unihandecode.`**`register_decoder`**(*lang*, *decoder*)

Registers a decoder (globally). The `lang` argument is the same that will be used in `unihandecode.Unihan()`. `decoder` should be a subclass of `unihandecode.BaseDecoder()`.

`unihandecode.`**`unregister_decoder`**(*lang*)

Unregisters a decoder (globally), this is mostly useful for testing.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*