Namerer Documentation

Release 0.6.7

Mitch Denny

Aug 28, 2018

Contents

Namerer is a cross-platform name text generator specifically designed to ease the process of coming up with unique company, product, brand or project names. Beyond just random strings, Namerer supports a template approach which allows you to zero in on the name that you want to use.

You can use Namerer directly on the command-line, or within your own Node.js projects.

CHAPTER 1

Getting Started

Getting started with Namerer is easy. You just need to make sure you have the right prerequisites installed, and then pull down the NPM package.

1.1 Prerequisites

Namerer requires Node.js 4.0.0 or greater to be installed, but you may as well just grab the latest version because that is what we build and test against.

1.2 Installation

Once you have Node.js installed and configured on your system, you just need to pull down and install the `namerer` NPM package using the following command:

```
npm install -g namerer
```

This will install the `namerer` package globally so that you can issue commands anywhere in the shell. Alternatively you can install it locally which is especially useful if you want to use it as a library for your own project.

1.3 Hello World

Once you've installed Namerer, it is time for a simple hello world example to make sure everything is working. The simplest command in Namerer is a basic `generate` command, invoked as follows:

```
$ namerer generate
```

This will output a single string which should be eight characters long comprised of characters from the alphabet, for example:

```
ighhkccy
```

Namerer is template driven so you can actually control what is output so you could make it output only four random characters by issuing the following command:

```
$ namerer generate "????"
```

You can learn more about how Namerer works in the *Generating Names* section.

CHAPTER 2

Generating Names

As demonstrated in the *Getting Started* section, generating a names using Namerer is as simple as using the `generate` command, for example:

```
$ namerer generate
```

You can control what Namerer generates by providing a template as an argument to the `generate` command. For example:

```
$ namerer generate "????"
```

This would generate a simple four character output string, for example:

```
yjrq
```

By default Namerer generates a single name, but you can use the `--count` option to generate more, for example:

```
$ namerer generate --count 5
```

This would generate something like the following output:

```
fkeyshtt
ytgebziv
kvitnilx
cvmmwvhz
tfsinukm
```

You can find out more about the various name generation command-line options in the *Command-line Options* section. The real power of Namerer however comes from the templates that you can provide.

2.1 Templates

A template is a string that you pass into the Namerer `generate` command which controls the shape of the name that is generated. Namerer provides a shortcut syntax for simple alpha and numeric which you can read about in the *Basic*

Templates section which expands into a JavaScript-powered function syntax that you can read about in the *Template Functions* section.

2.1.1 Basic Templates

The Namerer `generate` function uses templates to control its output. Templates are strings interspersed with placeholders which when processed are replaced with values which correspond to the kind of placeholders used. For example the basic template `???###` could be transformed into `abc123`.

The `?` and `#` placeholders are really just shorthand for a more function-based syntax. Prior to processing a template into a string the `?` and `#` characters are first converted into equivalent function-based syntax. For example, the `???###` template would be expanded into the following:

```
[alpha()][alpha()][alpha()][numeric()][numeric()[numeric()]
```

We'll explore *Template Functions* a little later, but for now know that `?` is the same as `[alpha()]` and `#` is the same as `[numeric()]`. At this point in time `?` and `#` are the only two shorthand characters. Now that you've got the basics you should check out the *Template Functions* section.

2.1.2 Template Functions

Template functions are the core of Namerer's string generation capabilities. A template function is a special token delimited by square brakets (for example `[vowel()]`) which when processed is replaced by a random string, the nature of which varies depending on which function you used.

Namerer has a bunch of different template functions from basic random character selection to syllable generation and synonym discovery. The following sections explain each of the template functions.

Usage of template functions is simple. The function, enclosed in its square brackets is placed in the template string passed into the `generate` function. The following invocation is an example:

```
$ namerer generate -c 5 "[syllable()]"
jwa
piv
wigh
wef
un
```

The `[syllable()]` function takes an optional boolean argument which specifies whether *phonetic vowels* can be used. Here is an example of the same invocation with the first argument (`usePhoneticVowels`) set to false. See how it affects the nature of the output:

```
$ namerer generate -c 5 "[syllable(false)]"
hu
wiw
ut
on
he
```

You can read up on the *[syllable(usePhoneticVowels?)]* in more detail in the following sections. You can combine template functions in a single template easily, for example:

```
$ namerer generate -c 5 "[syllable(false)][syllable()] [synonym('store')]"
qiug outlet
rodkah storage
```

(continues on next page)

(continued from previous page)

```
yisil outlet
esro depositary
qawlug depositary
```

It's really through combining multiple template functions together and adding in character sequences that you really want in the name that you find the usefulness of the Namerer tool.

2.1.2.1 [alpha(count?)]

The `[alpha(count?)]` template function outputs a random character constrained by the `--alphabet` option which can optionally be passed into the `generate` command. The function supports an optional `count` parameter which allows you to specify how many random alpha characters to output. The following table maps example templates to possible outputs.

Template	Output
[alpha()]	a
[alpha(1)]	w
[alpha(5)]	esome

2.1.2.2 [numeric(count?)]

The `[numeric(count?)]` template function is similar to the *[alpha(count?)]* in that it generates a random character, but it is instead constained by the `--numbers` option which can optionally be passed into the `generate` command. This function also supports an optional `count` parameter which allows you to specifiy how many random numeric characters to output. The following table maps example templates to possible outputs.

Template	Output
[numeric()]	3
[numeric(1)]	9
[numeric(5)]	31337

2.1.2.3 [vowel(count?)]

The `[alpha(count?)]` template function is similar to the *[alpha(count?)]* in that it generates a random character, however it will only generate a simple vowel, such as `a`, `e`, `i`, `o`, or `u`. This function also supports an optional `count` parameter which allows you to specify how many vowels to output. The following table maps templates to possible outputs.

Template	Output
[vowel()]	a
[vowel(1)]	e
[vowel(5)]	iouee

2.1.2.4 [phoneticVowel()]

The `[phoneticVowel()]` template function outputs a single string and takes no arguments. It outputs a vowels similar to the *[vowel(count?)]* template function, but also adds additional phonetic vowels. The following table shows

basic vowels and their related phonetic vowels that might also be output when using the `[phoneticVowel()]` function.

Basic Vowel	Phonetic Vowels
a	ai ay au aw augh wa all ald alk alm alt
e	ee ea eu ei ey ew eigh
i	ie ye igh ign ind
o	oo oa oe oi oy old olk olt oll ost ou ow
u	ue ui

2.1.2.5 [consonant(count?)]

The `[consonant(count?)]` template function is similar to the *[vowel(count?)]* in that it generates a random character, however it will only generate a consonant such as `b`, `c`, `d`, `f`, `g`, `h`, `j`, `k`, `l`, `m`, `n`, `p`, `q`, `r`, `s`, `t`, `v`, `w`, `x`, `y`, or `z`. This function also supports an optional `count` parameter which allows you to specify how many consonants to output. The following table maps templates to possible outputs.

Template	Output
[consonant()]	z
[consonant(1)]	b
[consonant(5)]	phjkl

2.1.2.6 [syllable(usePhoneticVowels?)]

The `[syllable(usePhoneticVowels?)]` template function is very useful for generating names which are easier to pronounce than random strings that might be generated by the *[alpha(count?)]* template function (for example). The `[syllable(usePhoneticVowels)]` function reuses the logic from the *[vowel(count?)]*, *[phoneticVowel()]* and *[consonant(count?)]* template functions. It randomly selects from the following four possible equivalent templates.

1. `[consonant()][vowel()]`
2. `[consonant()][vowel()][consonant()]`
3. `[vowel()][consonant()]`
4. `[consonant()][phoneticVowel()]`

The fourth option is included by default, however, it can be disabled when the `usePhoneticVowels` optional parameter is set to false. The following is an example of its usage:

```
$ namerer generate -c 5 "[syllable(false)][syllable()]"
xucpa
inkwa
kucta
etheigh
varom
```

2.1.2.7 [synonym(word)]

The `[synonym(word)]` template function can be used to find words which have a similar or related meaning to the value of the `word` parameter. The `[synonym(word)]` template function calls an external web service at Big Huge Labs. At this point in time the `[synonym(word)]` template function should be considered experimental and may

fail if it is used heavily across all Namerer users (because of API call limitations). The following is an example of its usage:

```
$ namerer generate -c 5 "[synonym('port')]###"
turn351
opening171
side462
porthole843
turn118
```

2.2 Command-line Options

You can display the list of command-line options for the `generate` command by adding a `--help` option to the command, for example:

```
$ namerer generate --help

  Usage: generate [options] [template]

  Options:

        -h, --help                 output usage information
        -a, --alphabet [alphabet]  Selection of letters to generate from.
        -n, --numbers [numbers]    Seletion of numbers to generate from.
        -c, --count [count]        Number of names to generate.
```

The `--alphabet` or `-a` option takes a list of characters and uses them to constrain which characters can be used when replacing a `?` token or `[alpha()]` function in the template string. For example, take the following command and its result:

```
$ namerer generate --alphabet abc "????"
acba
```

The `--numeric` or `-n` option works the same way, but instead controls what digits can be injected when the `#` token or `[numeric()]` function are used in the template string. For example you might want to append some digits to a product name but avoid what some cultures might consider to be unlucky numbers, for example:

```
$ namerer generate --numbers 0235789 "cafe ###"
cafe 203
```

Finally the `--count` or `-c` option takes a numeric value and controls how many instances of a particular template you want to generate:

```
$ namerer generate --count 5 "???###"
vyo148
xyx152
sqp102
apt577
njz132
```

That can be very useful when you want to generate some sample data, or just a selection of names to consider in one pass.

CHAPTER 3

Filtering Names

The Namerer `filter` command is a useful utility to take a name (or multiple names via stdin) and then check whether it is available. The current implementation of the `filter` command supports checking for DNS domain names with one more more suffixes. Refer to the *Command-line Options* section.

The `filter` command works by taking an input string (or multple) and then performing a number of availability checks. If the name passes all of the availability checks it is output to stdout. Here is an example of checking a single name:

```
$ namerer filter "somerandomname"
somerandomname
```

The above invocation worked because the `somerandomname.com` domain was available. In contrast, the following invocation would return nothing:

```
$ namerer filter "microsoft"
```

The Namerer `filter` command is designed to be used in conjunction with the `generate` command to quickly zero in on names that have a good chance of being usable. Here is an example of how you might use them to come up with a name with two syllables and check that the `.com` and `.io` suffixes are available:

```
$ namerer generate -c 10 "[syllable(false)][syllable()]" | namerer filter -d com,io
zetjim
amza
vogoy
viij
ufmall
halev
ozyun
nopoll
```

In this case only 8 names made it through the filter meaning that 2 of the 10 generated names had either a `.com` or `.io` suffix.

3.1 Command-line Options

You can display the list of command-line options for the `filter` command by adding a `--help` option to the command, for example:

```
$ namerer filter --help

Usage: filter [options] [name]

Options:

        -h, --help                  output usage information
        -d, --dnssuffixes [suffix]
```

The `--dnssuffixes` or `-d` option takes a comma-seperated list of DNS suffixes, for example:

```
$ namerer filter --dnssuffixes com,com.au "somerandomname"
somerandomname
```

If the `--dnssuffixes` option is excluded then the current behaviour is that a `.com` suffix will be assumed. In the future when future checks are performed for other services it may be that a bare `--dnssuffixes` option will apply `.com` and other sensible defaults and its absence will skip the DNS check altogether.