Placekey Encoding Specification

This document provides a specification for the encoding scheme used for Placekey. For a broader technical introduction to Plackey see the Placekey Technical White Paper. For a reference implementation of the encoding described in this document see the placekey-py repository.

For the purposes of what follows recall that a Placekey is formatted:

Placekey Integer Encoding

The plackey encoding algorithm operates in two steps:

1. Convert the integer into the encoding alphabet,
2. Replace, in order, each avoided strings with its clean variant.

Decoding an encoded integer reverses these steps:

1. Replace, in reverse order, the clean variant of each avoided string with its corresponding avoided string,
2. Convert the string in the encoding alphabet into an integer.

Encoding Alphabet

Placekeys are encoded with the following 28 character alphabet:

23456789bcdfghjkmnpqrstvwxyz

The following characters are also reserved as special characters:

a e u

Encoding an integer into this alphabet is simply a base change from whatever base the integer was originally represented in to base 28, and vice-versa for decoding.

Avoided Strings and Their Replacements

Avoided strings have their terminal character replaced by one of the special characters eu. The character e is used unless that could introduce another avoided string, and in that case u is used instead.

The list of avoided strings and their replacements is ordered to handle cases where two avoided strings overlap (e.g., “prn” and “ngr” overlapping as “prngr” will be transformed to “pregr” since “prn” is earlier in the list).

Num.	Avoided String	Clean Variant	Related Avoided String
0	prn	pre	
1	f4nny	f4nne	
2	tw4t	tw4e	
3	ngr	ngu	gey
4	dck	dce	
5	vjn	vju	jew
6	fck	fce	
7	pns	pne	
8	sht	she	
9	kkk	kke	
10	fgt	fgu	gey
11	dyk	dye	
12	bch	bce	

What Parts

The What part of a Placekey has two parts, each a three character encoded integer:

1. Address encoding (required)
2. POI encoding (optional)

The character **2** is used to left pad the encoded value, e.g., the encoding of 0 is "222" and the encoding of 10 is "22c". Note that are 28^3 = 21,952 possible values for the address and POI encodings.

Case	What part format	Example
Address only	{address-encoding}@	2b4@
Address and POI	{address-encoding}-{poi-encoding}@	2b4-227@

Address and POI encodings are assigned incrementally and hierarchically

1. Address encodings are assigned serially per Where part
2. POI encodings are assigned serially per Address@Where

Case	Example Placekey
Where only	@5vg-7gq-tvz
1st address at Where	222@5vg-7gq-tvz
10th address at Where	22c@5vg-7gq-tvz
1st POI at 1st address at Where	222-222@5vg-7gq-tvz
5th POI at 1st address at Where	222-226@5vg-7gq-tvz
5th POI at 10th address at Where	22c-226@5vg-7gq-tvz

There are a handful of reserved address encodings related to the CASS validity of the address. A CASS validated address is one to which the USPS can deliver mail.

Reserved address encoding	Use
zzz	An addressable location without a CASS-valid mailing address, e.g,. a park
zzy	A CASS-invalid address for which SafeGraph has a POI
zzw	A CASS-valid address

Where Parts

The Where part of a Placekey is a 9 character encoding of a modified H3 index of resolution at most 12. Currently, only H3 indices of resolution 10 are used for Placekey. The integer value of the H3 index is modified so that fewer bits (and therefore fewer characters) are required to represent the index and for comparability of encoded strings.

When the value encoded in a Where part has length less than 9, the character a is used to left pad the string to length 9.

H3 Index Truncation

Below is the bit layout of a 64-bit integer that is an H3 index. The 43 bits used for a Where part are highlighted in blue.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Res	Mode				Res./edge			Resolution				Base cell			
2	Base cell				Digit 1			Digit 2			Digit 3			Digit 4		
3	Digit 5			Digit 6			Digit 7			Digit 8				Digit 9		
4		Digit 11			Digit 12			Digit 13			Digit 14			Digit 15		

These bits can be truncated without loss of information because they are either constant or their value can be inferred from the remaining bits (in the case of resolution).

Bits	Reason for truncation
1	This bit is reserved in H3 and always set to **0b0**.
2-5	These indicate the mode of the H3 index, and are set to **0b0001** when indexing hexagonal cells
6-8	These bits are reserved in H3 and always set to **0b000**.
9-12	"These bits encode the resolution of the H3 index. When the index is resolution 10 they are **0b1010**. Resolution can be inferred from the number of digits in the bit layout which are not **0b111**, as the digit value can only be 0b111 when that digit is greater than the resolution of the index."
56-58	These bits encode resolution 13. If the H3 index has resolution at most 12, these are **0b111**.
59-61	These bits encode resolution 14. If the H3 index has resolution at most 12, these are **0b111**.
62-64	These bits encode resolution 15. If the H3 index has resolution at most 12, these are **0b111**.

Base Cell Index Shift

Base cells in an H3 index are valued between 0 (**0b0000000**) and 121 (**0b1111001**). Since the value for the base cell and all subsequent digits can be 0, this means that once the leading 12 bits of the index are truncated we might be encoding the integer 0. Requiring 8 padding characters is undesirable so we increment the value of each base cell by 1. By doing this, only a single padding character is ever needed for any Where part.

H3 Encoding

The full process for generating a Where part given an H3 index is

1. Increment the value of the base cell by 1, equivalent to adding 2^{45} to the H3 index,
2. Remove bits 1-12, equivalent to finding the remainder mod 2^{52},
3. Remove bits 56-64, equivalent to integer division by 2^{9},
4. Encode the value in the encoding alphabet,
5. Left pad the encoded string with “a” until it is length 9.

The decoding process inverts the above steps

1. Remove any “a”s from the encoded string,
2. Decode the string into a binary value,
3. Restore bits 56-64, equivalent to adding **0b111111111**,
4. Determine the resolution of the value by finding the largest digit not filled with **0b111**,
    - In practice this will be the 10th digit.
5. Compute the corresponding header bits based on the resolution, multiply by 2^{52}, and add to the value,
    - For resolution 10 these bits are **0b10001010**, and the value to be added is 621496748577128448 in base 10.
6. Decrement the value of the vase cell by 1, equivalent to subtracting 2^{45}.