Twitter Thread by Sam Nicholls

Sam Nicholls
@samstudio8

I have distracted myself this week with the extraordinarily mundane task of designing a sample identifier scheme. I want to share some decisions I made in the hope that it saves somebody else some time.

The first choice was the alphabet. I wanted to use human friendly, familiar ASCII characters, but intentionally leave out potentially confusing characters. I'd previously read about "Crockford's base32" (https://t.co/xa3WREc1RQ) which tries to address this problem.

But my search led me instead to z-base32 (https://t.co/5PlTgAgyDU). The z-base32 alphabet is shuffled to try and make encoded identifiers easier to discern. I don't actually need the encoding, but I liked the idea that this shuffle makes sequential identifiers less sequential.

z-base32 takes the interesting choice of using the letters in lowercase to help identifiers form "coastlines" that aid with human recognition. The lab quickly fed back they didn't like this, so I force the alphabet back to uppercase.

With the alphabet decided, I wanted to pick a checking scheme. I learned that each algorithm catches different types of errors, so one needs some knowledge of how the identifiers will be used when making a decision.

Some simple algorithms will catch single symbol mistakes, but not transpositions of adjacent symbols - which seems inappropriate for detecting transcribing problems in laboratories. I chose Luhn mod N, which is just an extension of the Luhn checksum to alphabets of length N.

Next I wanted to try and address some problems I had seen in sample identifiers before; repeated characters that need to be counted (e.g. 000001), similar looking characters that often get flipped by your brain (83, D0) and characters that get mistaken for each other (0D).

The third problem is already somewhat addressed as z-base32 removes 0 (zero), 2 (two), l (lima) and v (victor) from the alphabet, but we can do better to avoid mistakes.

After some searching, I found a great resource from the "Institute for Safe Medication Practices" which tabulates some commonly confused symbol pairs along with some quite horrific war stories about what happens when doctors and nurses

confuse such symbols
https://t.co/DAms8Jj6A7

I also discovered that some medications have such similar names that a scheme called "Tall Man lettering" (where part of the drug name is in uppercase: eg. buPROPion vs. busPIRone) was introduced. The ISMP also maintains this list.
https://t.co/T2hKbvpCPy

Although the ISMP article seems to focus on handwritten symbols, many of the entries in the table are just as applicable to printed characters. To help reduce confusion in my scheme, I took all the pairs in this table and ban any identifier that contains even just one.

Handling repeats is straightforward. We just decide how many times a character can appear in a repeat and ban any identifiers that contain a run of symbols greater than that. Somewhat arbitrarily I picked two, as in my experience people seem to read identifiers in pairs.

So we have an alphabet, a check algorithm, a way to mitigate repeats, confusing symbols and commonly flipped symbol pairs. Finally I needed to choose how long the identifiers should be.

The identifiers need to be long enough to cover a large search space to cover our needs for the forseeable future, but not so long that they would become hard to read. base32 helps us because we can fit more combinations in the same number of chars than say numbers only, or hex.

With four identifiers we have a theoretical maximum of 32^4 which is a little over a million. This seems more than enough at first thought, but I assumed that if I did such a good job that other groups may want to borrow our scheme.

Within @CovidGenomicsUK we are rapidly exhausting the hex-space, and uploaded over 100,000 new identifiers in the past two months. If someone decided to deploy this scheme to the whole project, our 32^4 space would be depleted by 2022.

32^5 offers a theoretical maximum of over 33 million, which is hopefully a millions more than we will ever need. At 32^5 the space is large enough to be divided into huge allocations that can be assigned to different organisations and projects without running out (remember IPv4)

I say theoretical of course because we'll lose large pieces of the identifier space by banning repeats and confusing flips (e.g. DO*** has 32k, AAA** has 1024). Cutting 25% of the space would yield 25 million. Even at 50% we'd have around 16 million identifiers to play with!

Of course this is all lovely thoughts but it's pointless without a robust way of generating such identifiers. So I started writing some software. Naaavi is a tool for "naming avoiding ambiguity automatically assigning valid identifiers".
https://t.co/qIiRWEZxBP

For a given alphabet (z-base32) and size (5), Naaavi will exhaustively iterate through the identifier space. Each identifier is then checked against a set of enabled "rejectors", if the identifier fails a rejector, we move on, if it doesn't, it goes to stdout.

I've also added the `better_profanity` Python package as a rejector to try and avoid generating amusing samples identifiers, because it seems unbecoming to upload samples named FUCKIT0 to public databases.

I've also added a `not_exclusive` rejector to prevent generating identifiers composed exclusively of a given character set. I use this to prevent generating an identifier composed only of hex chars that could potentially be confused with the ones @CovidGenomicsUK still uses.

To avoid confusion between the Naaavi software and the scheme I propose, I have named this scheme "Zeal-5". Zeal as in "a great energy" or "a herd of zebra". The first two of the five characters are used to split the space into large groups (775k) and smaller subgroups (25k).

Zeal identifier layout

BIRM-	B	E	SAM	5
prefix site/project identifier alphanumeric then -	**group** first collision space z-base-32	**subgroup** second collision space z-base-32	**name** identifying code z-base-32	**check digit** Luhn mod N validator z-base-32

- A `prefix` will hold approximately 25 million identifiers so you'll run out of labels before identifiers
- A `group` holds approximately 775,000 identifiers, a `subgroup` approximately 25,000
- The `group` and `subgroup` are part of the identifier and must always be provided to avoid ambiguity
- Likewise, the `check digit` is only used to validate the identifier, but must always be provided
- An identifier requires all of these pieces to be valid and unambiguous, e.g. **`BIRM-BESAM5`**

I have always had strong opinions about sample identifiers and feel like I've finally put my money where my mouth is in designing this scheme. We're about to deploy these to the lab so I'll keep this thread updated with any additional findings. I'd love to hear any thoughts!

I plan to add more alphabets and check digit functions to Naaavi in the future. I especially want to come back to my idea of using the "diceware" password methodology to generate triples of words to name boxes in the freezer https://t.co/8vng8TW3Hf

> this would never have happened with our diceware labels pic.twitter.com/6JYbVk70LR
>
> — Sam Nicholls (@samstudio8) February 12, 2020

I provide Naaaavi and these thoughts in the hope that it might be useful to someone in future, and not necessarily as the perfect system to deploy outright. Many decisions (alphabet, symbols to remove, flips to ignore, length) come down to how you intend to use these identifiers.

Clearly it's important to get sample identifier naming schemes right, just look at the grim examples from the ISMP for what happens when letters get mixed up.

ps. If you're in @CovidGenomicsUK and want to use these, please get in touch first. We'll have to co-ordinate ourselves and allocate blocks inside Zeal5 space!