Possible data storage and transmission formats

(anonymized ProdBase extracts)

Leo Heska
July, 2012
- - draft - -

There are a lot of ways we may store and transmit anonymized ProdBase extracts. Some require only brief mention:

- XML. Very readable but kind of passe, and bloated. The bang for us is small and the buck (processing overhead, bloat) too high.
- JSON. Favored nowadays; created as a way to reduce XML's bloat. It does, but only by about 50%. Still too bloated for our use.
- EDI and other proprietary, obscure, and or "technical" formats. Too much work and complexity; too-low bang for too-high buck.

Possible/candidate storage and transmission format we may use for anonymized ProdBase extracts:

- Key-value pairs
- Delimited flat files without header records
- Delimited flat files with header records
- Positionally formatted flat files, no header records
- Hybrid; positionally formatted followed by key-value
- Encoded single values

Advantages, disadvantages, and storage considerations of each of these, I present below for discussion and consideration.

Key-value pairs

Currently one of the (leading) options for formatting an anonymized ProdBase extract looks like this (lines truncated here for display):

```
0000008F91D0401CAF5BC063F29310C02E40D89F|1270-11|1271-08X|1274-08O|1275-03O|1280-10|1281-02H|4151-44|4152-2
000002A365739E99657CD84AB646637A9E5B66C9|1270-36|1271-20S|1274-07C|1275-04C|1280-45|1281-10L|4151-70|4152-1
000002F7994DF919B73F4ACFB0AF659CD5C0A3E7|1270-20|1271-07X|1275-03O|1280-26|1281-06M|2507-00|2531-00
```

Advantages:

- No header record needed as long as all parties recognize that the left column is an AshID. For humans that's easy - the AshID is the left column, it looks different fromn all the rest, it's the only column that doesn't contain a dash. These are probably distinction enough to "set it off" - humans can read this.
- Key-value pairs make the format self-documenting.
- Key-value pairs mean that new keys may be added at any time
  - No recoding of export or import processes needed
  - Downstream processes, though, may need recoding and/or reconfiguration, to make use of the new data.

Disadvantages:

- This hybrid format will never be maximally nice to ingest with automated tools. It's almost key-value pairs throughout, but not quite; the first field has no key. Therefore a "programming trick" will be needed prior to ingestion, if automated parsing tools that expect key-value pairs are to be used. Such "tricks" may or may not be available in all input processing routines, packages, and/or languages. This extra complexity can cause extra development effort, require more moving parts, and/or slow processing in production.
- Possibility of duplication
  - There is nothing within the file structure itself that prevents something like this:

```
000002F7994DF919B73F4ACFB0AF659CD5C0A3E7|1270-20|1271-07X|1270-3O
```

Bloat:

- At a glance at sample data, about 65% bloat:
  - Other than the AshID, every field consists of 1 to 3 characters of information, but 6 additional characters are required for the encoding. Like so:
    - Field: `|1270-11`
    - Information content: `11`
    - Formatting and metadata: `|1270-`
    - Average content, 2 characters. Average formatting and metadata, 6 characters. This yields the 300% bloat.
- Upon analysis of sample data, 81% bloat:
    - 72,651 bytes of values (50,760 total values)
    - 381,091 bytes total of all value strings
    - This yields the 81% bloat

Delimited flat-file records without headers

Another option, apparently currently possible/available (the data feed already exists), looks like this (lines truncated here for display):

```
0000008F91D0401CAF5BC063F29310C02E40D89F|113|08X3|08O3|03O3|101|02H1|44|2|Y|000000001001000|M|O|01|S||||||||||1|||||||
000002A365739E99657CD84AB646637A9E5B66C9|363|20S3|07C3|04C3|451|10L1|70|1|Y|001000000000000|M|R|03|M|1|1|||||||||1||||
000002F7994DF919B73F4ACFB0AF659CD5C0A3E7|20G|07XG||03OG|26G|06MG|||000000000000000|||||||||||||||||||||||||||
```

Advantages:

- No "programming trick" needed to ingest. All fields are just fields. Very many input routines, tools, and packages can handle this format as is.
- If the ingestor is configurable:
  - No recoding of export or import processes needed
  - Downstream processes, though, may need recoding and/or reconfiguration, to make use of the new data.
- No possibility of duplication as is possible with key/value pairs; this format does not permit transmitting values that conflict.

Disadvantages:

- This format is not self-documenting. To ingest the data, a separate metadata store must be communicated to the ingesting program, either at design/coding or configuration time. The possibility does exist that the data layout may change, but the metadata will not, creating an out-of-synchronization condition.

Bloat:

- 5% improvement over key/value pairs.
- Sample data show that average line length of records stored in this format is 404 characters, versus 421 characters formatted as key/value pairs.
- The 210 delimiters per line are "waste" from an information content point of view. However, this is more than offset by eliminating the verbose keys.
- Internal bloat shows 48% data, 52% delimiters.

Delimited flat-file records with headers

Another option would look like this (lines truncated here for display):

```
AshID|FieldName1|FieldName2|FieldName3|FieldName4|FieldName5|FieldName6|FieldName7|FieldName8|FieldName9|FieldName10|
0000008F91D0401CAF5BC063F29310C02E40D89F|113|08X3|08O3|03O3|101|02H1|44|2|Y|000000001001000|M|O|01|S||||||||||1|||||||
000002A365739E99657CD84AB646637A9E5B66C9|363|20S3|07C3|04C3|451|10L1|70|1|Y|001000000000000|M|R|03|M|1|1|||||||||1||||
000002F7994DF919B73F4ACFB0AF659CD5C0A3E7|20G|07XG||03OG|26G|06MG|||000000000000000|||||||||||||||||||||||||||
```

Advantages:

- Some input routines, tools, and packages can handle this format as is (but see related disadvantage below).
- If the ingestor is configurable:
    - No recoding of export or import processes needed
    - Downstream processes, though, may need recoding and/or reconfiguration, to make use of the new data.
- No possibility of duplication as above; this format does not permit transmitting values that conflict.
- This format is self-documenting. Metadata are included along with the data. Though there is a possibility that a developer would change data columns without changing the header records, this possibility is relatively slight.

Disadvantages:

- Some input routines, tools, and packages can not handle this format as is, necessitating pre-processing and/or error handling to pull of and/or discard the header record.

Bloat:

- Effectively the same as flat-file records without headers; all we do is add one header record; a tiny relative addition in the case of large files.

Positionally formatted flat files, no header records

Consider the following; a portion of a data feed that currently exists (records truncated):

```
0000008F91D0401CAF5BC063F29310C02E40D89F|113|08X3|08O3|03O3|101|02H1|44|2|Y|000000001001000|M|O|01|S||||||||1
000002A365739E99657CD84AB646637A9E5B66C9|363|20S3|07C3|04C3|451|10L1|70|1|Y|001000000000000|M|R|03|M|1|1||||||1
```

Note how nicely the fields all line up. Examining the total sample input data file, they don't all line up as perfectly as that, but the sameness of format and occurrence of values is apparent. In such a situation, the following data formatting option may be used:

```
0000008F91D0401CAF5BC063F29310C02E40D89F11308X308O303O310102H1442Y000000001001000MO01S     1
000002A365739E99657CD84AB646637A9E5B66C936320S307C304C345110L1701Y001000000000000MR03M11   1
```

Advantages:

- This is, or at least may be, the most efficient of all our options. The overhead of using spaces when particular data values are not present, is minimal.
- Maximally easy to ingest using nearly any off-the shelf/existing input routine.

Disadvantages:

- This looks old-fashioned! Though seemingly silly, we must not discount this factor. This scheme predates both computers and electronics, going all the way back to the Hollerith card and the 1890 (no that is not a typo) U.S. census. No one will ever look at a presentation of this scheme and react "cool!" Even if it happens to be the best option, all things considered.
- As with other headerless formats, the possibility does and always will exist that column layouts could change, without correspondingly changed metadata being transmitted to/incorporated by the ingesting system.

Bloat:

- Possibly, near-zero - close to "as small as bloat can get". But see also hybrid schemes.

Hybrid: Positionally formatted flat files followed by key-value pairs (no header records)

Consider the following (extracted from actual sample data but modified and edited, for length and to demonstrate the point):

```
00000AF26FC46AF0613133E783A99AC4F7E880BD|173|12B3|01C3|01C3|261|06M1|34|1|Y|000001000000000|M|O|01|S||1|1||1||||G|B|C|C|||G|||A||M|||||000000010000010000000|byr=1941|
00000C3FBE1125E0273D8C06C5F1B52DE722F82B|05G|19MG|    |02OG|28G|03MG|  | | |000000000000000| | |  | || | || |||| | | | ||| ||| ||
|||||000000000000000000000||
00000D35294BA9AC67E18F86174D08DD61E1639D|20G|07XG|12OG|03OG|18G|03MG|  | | |000000000000000| | |  |S|| | || |||| | | | ||| ||| ||
|||||000000000000000000000||
```

Again, we may advantageously remove delimiters, going to a positional scheme. But note toward the end of the top line, the value 1941, not present in other records. If there are enough instances of this sort of thing (variant attribute availability) then a hybrid scheme might be optimal. First, on the left side (before the vertical bar character), come all the positionally formatted data. Then come the key-value pairs. Like so:

```
00000AF26FC46AF0613133E783A99AC4F7E880BD17312B301C301C326106M1341Y000001000000000MO01S111GBCCGAM000000010000010000000|byr=1941
00000C3FBE1125E0273D8C06C5F1B52DE722F82B05G19MG    02OG28G03MG    000000000000000               000000000000000000000
00000D35294BA9AC67E18F86174D08DD61E1639D20G07XG12OG03OG18G03MG    000000000000000    S          000000000000000000000
```

Advantages:

- If ProdBase records consist of a mix of attributes that are:
  - "very commonly present" and
  - "sometimes present but with great variation,"

  then this would be the most efficient of all our options. We use the best of both worlds; skipping the overhead of both delimiters and of keys when that is beneficial, but then, using keys when (and only when) it is beneficial to do so.

Disadvantages:

- Custom input/parsing required. No off-the-shelf tool I know of supports this.
- Metadata still required for the left hand data elements; out-of-synch potential exists. As with other headerless formats, the possibility/risk will always exist, that column layouts could change, without correspondingly changed metadata being transmitted to/incorporated by the ingesting system.

Bloat:

- Possibly, near-zero - close to "as small as bloat can get".

Encoded single values

Looking again at one of the (leading) options for formatting an anonymized ProdBase extract (lines truncated here for display):

```
0000008F91D0401CAF5BC063F29310C02E40D89F|1270-11|1271-08X|1274-08O|1275-03O|1280-10|1281-02H|4151-44|4152-2
000002A365739E99657CD84AB646637A9E5B66C9|1270-36|1271-20S|1274-07C|1275-04C|1280-45|1281-10L|4151-70|4152-1
000002F7994DF919B73F4ACFB0AF659CD5C0A3E7|1270-20|1271-07X|1275-03O|1280-26|1281-06M|2507-00|2531-00
```

There is another scheme, that is already used at Anonom and could be used in this instance. That is:

- encode all this information as single values
- each single value contains both a definitional and a data element.

So, working from the above example:

- `aaaaa means 1270-11`
- `aaaab means 1270-20`
- `aaaac means 1270-36`
- `aaaad means 1271-08X`
- `aaaae means 1271-20S`
- and so on.

Obviously this is just a "toy" example and the real scheme would use different mappings. But as long as the ProdBase information to be stored and transmitted consists of a smallish set of enumerable values (which would exclude storing information such as decimal values), this scheme is possible.

Advantages:

- Variable-length records nicely supported.
- Order of data within a record is not important.
- New encodings may be added at any time
    - No recoding of export or import processes needed
    - Downstream processes, though, may need recoding and/or reconfiguration, to make use of the new data.

Disadvantages:

- This scheme does still inhere the risk of non-synchronization of metatdata and data. If the definitional scheme ever changes, but that is not communicated to the ingesting system, incorrect processing and interpretation will result.
- Possibility of duplication
    - There is nothing within the file structure itself that prevents something like this (given the sample scheme shown above, this is an internal contradiction):

      ```
      000002F7994DF919B73F4ACFB0AF659CD5C0A3E7|aaaab|aaaac|
      ```

Bloat:

- This sort of scheme will almost certainly not be efficient as regards data storage, compared to some others described here. In the example shown here, using 5 alphanumeric characters (which seems a plausible length), it therefore takes 5 characters to express what is expressed in just one character in some of the other schemes. Which works out to 80% bloat. A better analysis, using sample data, I have not carried out.