**SRS Substance hash algorithm**

**Background & Terminology:**

***MD5 Hash:*** A popular hashing algorithm, with widespread availability. This hash algorithm is quick, fairly simple, and is standard in most programming languages. It is common to use the MD5 hash of large byte arrays (e.g. large files) to confirm integrity or identity without having to do bitwise comparisons, as very small differences in a source array still tend to result in completely different hashes. <https://en.wikipedia.org/wiki/MD5>

***Note:*** *The MD5 hash is technically a byte-array which is deterministically produced from a different byte array. However, in this document, “MD5 hash” will typically refer to a hexadecimal character string representation of the resulting byte array. Similarly, “producing an MD5 hash” for a String array typically means encoding the original string as an ASCII byte array, producing the MD5 hash, and then encoding that hash as a hexadecimal character string.*

**Introduction**

In order to detect sufficient uniqueness of substance definitions, SPL substance indexing files employ the use of a hashing algorithm. Using this algorithm, an SPL substance definitional elements can be meaningfully encoded as a 128-bit (32 ASCII HEX character) string. If properly encoded, the expectation is that two sufficiently identical substances will receive the same hash string, while 2 non-identical substances will have a *very* low (but not zero!) probability of receiving the same hash. The procedure for generating such a hash is necessarily complex for the various substance classes. This document serves as a guide to how this process is achieved.

Basic procedure:

1. Receive substance document (SPL or other)
2. Extract all elements which are relevant to the definition
3. Canonicalize each element to deterministic string, reuse existing libraries as much as possible (e.g. InChI)
4. Combine all elements in a canonical way (e.g. concatenate strings with delimiters)
5. Produce MD5 hash of resulting deterministic strings

Note that this process often involves “hashes-of-hashes”. That is, the deterministic representation of sub elements may also involve a hash function.

Consider the example of a protein substance.

For a Protein Substance, we consider the following to be definitional:

* Subunits
* Links
* Glycosylation
* Modifications
* Fragments

The hash for a protein substance is assembled by the hashes of these individual elements. For this approach to work, each element must be:

1. Sufficiently independent of other elements
2. Deterministic in its definition
3. Deterministically ordered

To create *the* canonical hash for the full protein substance, we first concatenate the hashes of each of the elements in a canonical order. This produces a large string, with the “\_” character used to separate element types, and the “|” character used to separate elements within a type.

See the code below to see how this is done:

public override string UID

{

get

{

return (

String.Join("|", Subunits.OrderBy(s => s.Sequence.ToString())

.Select(s => s.UID)) + "\_" +

String.Join("|", Links.Select(s => s.UID)) + "\_" +

String.Join("|", Glycosylations.Select(s => s.UID)) + "\_" +

String.Join("|", Modifications.Select(s => s.UID)) + "\_" +

String.Join("|", Fragments.Select(s => s.UID))

)

.GetMD5String();

}

}

For a protein substance, the algorithm is as follows:

1. For each subunit create canonical representation (e.g. choose “alphanumerically lower” sequence of characters comprised from direct and reverse sequences)
2. Order all *Subunit*s alphabetically by sequence
3. For each *Subunit*, determine its hash\*
4. Concatenate all *Subunit* hashes, in order, separating by “|”
5. Then add each *Link*’s hash, ordered as they appear in the SPL document, and concatenate them together separated by “|” as well, append that string to the running total string with a “\_” separator.
6. Then add each *Glycosylation’*s hash, ordered as they appear in the SPL document, and concatenate them together separated by “|” as well, append that string to the running total string with a “\_” separator.
7. Then add each *Modification’s* hash, ordered as they appear in the SPL document, and concatenate them together separated by “|” as well, append that string to the running total string with a “\_” separator.
8. Then add each Fragment*’s* hash, ordered as they appear in the SPL document, and concatenate them together separated by “|” as well, append that string to the running total string with a “\_” separator.
9. Finally, take the whole resulting string and return the MD5 hash.

A more detailed breakdown of each element’s hash (often recursive) can be found below:

**Breakdown of Hashes**

***Protein Hash***

*Dependencies*

* Forced-order *Subunit* hashes (sorted alphabetically by sequence)
* Given-order *Link* hashes
* Given-order *Glycosylation* hashes
* Given-order *Modification* hashes
* Given-order *Fragment* hashes

*Notably Not Used*

* Protein Type
* Sequence Type

*Algorithm*

* Concatenate the above hashes, in the order stated, separating each element in a list with “|” and separate each list with “\_”
* Return the MD5 hash of the resulting string sequence

***Subunit Hash***

*Dependencies*

* Sequence string of amino acids

*Algorithm*

* Return the MD5 hash of the amino acid sequence string

***Link Hash***

*Dependencies*

* Linker fragment hash (optional)
* Forced-order *Site* hashes (sorted alphabetically by string notation)

*Algorithm*

* If there is no Linker fragment specified, return "<cannot be generated>" (not technically a hash, but still a deterministic string)
* Otherwise, concatenate the Site list hashes, separating each element with “-”, followed by another “-”, and the hash for the Linker fragment.
* ***Note***: This resulting string is *not* MD5 hashed.

***Fragment Hash***

*Dependencies*

* InChIKey of structure fragment
* Connector-pair atom indices (expected to be in InChI atom canonical order).

*Algorithm*

* First, add the InChIKey of the fragment to a running string
* Then, for each connector-pair of atoms (if there are connectors) add “-”, followed by the atom indices in the format “X\_Y”, joined by “-” for each connector pair
* Return the resulting string
* ***Note***: This resulting string is *not* MD5 hashed.

***Site Hash***

*Dependencies*

* Subunit index
* Residue index

*Algorithm*

* Return the site in the form “{Subunit\_index}\_{Residue\_index}” [1-indexed]
* ***Note***: This resulting string is *not* MD5 hashed.

***Glycosylation Hash***

*Dependencies*

* Glycosylation Type [string]
* ProteinAttachment Hash

*Algorithm*

* Return Glycosylation Type [string] followed by “-” and then the ProteinAttachment Hash
* ***Note***: This resulting string is *not* MD5 hashed.

***ProteinAttachment Hash***

*Dependencies*

* Attachment Type [string] {I’m not sure I understand this one}
* Site Hash

*Algorithm*

* Return Site Hash followed by “-” and then the Attachment Type string
* ***Note***: This resulting string is *not* MD5 hashed.

***Amount Hash***

*Dependencies*

* Numerator [double]
* High [double]
* Low [double]
* Unit [string] {defaults to “1”}
* Amount Type [string] {defaults to “EXACT”}

*Algorithm*

* Build a string representing the numerical portion as follows **{note that the precision of the string is not explicitly set}**:
  + If Numerator exists, return “<{NUMERATOR}>” (e.g. “<4>”)
  + Otherwise, if High and Low both exist, return “[{LOW}..{HIGH}]” (e.g. “[1..2]”)
  + Otherwise, if *only* High exists, return “[..{HIGH}]” (e.g. “[..2]”)
  + Otherwise, if *only* Low exists, return “[{LOW}..]” (e.g. “[1..]”)
  + Otherwise, if none of the three exist, return “”
* Taking the numeric portion above, construct a string of the following format
  + “{Amount Type} = {NUMERIC VALUE} {UNIT}”
  + (e.g. “EXACT = <5.2> 1” or “EXACT = [1..2] g/Mol”)
* Return the constructed string
* ***Note***: This resulting string is *not* MD5 hashed.