**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

Defined in Subunit.cs as follows:

public string UID

{

get { return Sequence.ToString().GetMD5String(); }

}

Sequence.ToString:

public override string ToString()

{

return \_sequence;

}

***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.

Defined in Link.cs as follows:

public string UID

{

get

{

return Linker == null ? "<cannot be generated>" : String.Join("-", Sites.OrderBy(s => s.ToString()).Select(s => s.UID)) + "-" + Linker.UID;

}

}

Example of a string that the above code returns:

SU1\_14-SU1\_79-<inchikey>-7\_5-8\_6

Green part represents the UID of the connection sites.

Blue part represents the UID of the linker, or in other words fragment that serves as a linker. This is analogous to Fragment.UID (see Fragment Hash section below)

Where:

SU1 is a subunit

SU1\_14 is a first site (residue 14 on subunit SU1)

SU1\_79 is a second site (residue 79 on subunit SU1)

Note that residues in this context are zero based, so first residue is 0, thus residue 14 is really a 15th residue in sequence)

<inchikey> of the fragment that constitutes the linker

7\_5 and 8\_6 are the linker connection points

***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.

Defined in Glycosylation.cs as follows:

public string UID

{

get { return GlycosylationType + "-" + Attachment.UID; }

}

Where:

GlycosylationType is a string (“HUMAN TYPE GLYCAN”, etc)

Attachment.UID is a ProteinAttachment.UID:

public string UID

{

get { return Site.UID + "-" + AttachmentType; }

}

Where Site.UID is ProteinSite.UID which returns ProteinStie.ToString():

public override string ToString()

{

return String.Format("{0}\_{1}", Subunit, Position);

}

Example: SU1\_14 (See protein links hash code for explanation)

AttachmentType that is used in ProteinAttachment.UID is a type of atom where attachment happens and is just a string (such as “C”,”O”, “N”) to specify a type of glycosylation attachment such as C-glycosylation or O-glycosylation. **NOTE: This information is not contained in the SPL.**

***Modifications Hash***

Defined in ProteinModification.cs as follows:

public virtual string UID

{

get {

return (

ModificationType +

Role +

Amount.UID

).GetMD5String();

}

}

Where:

ModificationType (string)

Role (string)

Amount – see below for Amount Hash

YP: I haven’t seen any examples of proteins with modifications, nor can I see SPL subroutine for the modifications in the code, so at this time I cannot comment beyond the above code snippet and brief explanation of components.

***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.

***-------------------------------------------------------------------------------------------------------------------------------***

***Determination of the lexicographical order of protein chains based on SPL Index information.***

***Amino acid (AA ) sequence is represented by single letter notation. I would not recommend splitting AA sequence into substrings and using InChI for substrings because such algorithm would be poorly reproducible and the hash wouldn’t be computable from SPL Index information alone.***

***Lexicographical order is determined in two iterations***

1. ***First iteration***

***Each modified AA is replaced by combination of its InChI and its connection points. All the info is explicitly present in the SPL Index file. For example, in file bf170df7-8153-45c1-9a21-6e1aa1b4c97e.xml,***

***amino acid SU3\_1 can be replaced by the InChI of the pyroglutamic acid: InChI=1S/C5H7NO3/c7-4-2-1-3(6-4)5(8)9/h3H,1-2H2,(H,6,7)(H,8,9)/t3-/m0/s1 combined with connection atoms [6,5]. The resultant string may be represented as [ InChI=1S/C5H7NO3/c7-4-2-1-3(6-4)5(8)9/h3H,1-2H2,(H,6,7)(H,8,9)/t3-/m0/s1]|[6,5] or likewise.***

***Each cysteine that forms a disulfide bond can be replaced by the InChI of the disulfide linker: InChI=1S/C6H12N2O4S2/c7-3(5(9)10)1-13-14-2-4(8)6(11)12/h3-4H,1-2,7-8H2,(H,9,10)(H,11,12)/t3-,4-/m0/s1 combined with connection atoms: (7,5;8,6). The resultant string may be represented as [InChI=1S/C6H12N2O4S2/c7-3(5(9)10)1-13-14-2-4(8)6(11)12/h3-4H,1-2,7-8H2,(H,9,10)(H,11,12)/t3-,4-/m0/s1 ]|[7,5;8,6] or likewise.***

***After concatenation of single letter notation and the strings that represent modified AAs, the AA sequences can be ordered in a lexicographical order and assigned ranks. It is possible that two or more sequences have the same rank.***

1. ***Second iteration***

***Each modification of type “link” is extended by adding the rank of the aa sequence and position of other amino acids that form that link. If the second aa belongs to the same sequence the rank is skipped.***

***After concatenation of single letter notation and the extended strings that represent modified AAs, the AA sequences can be ordered in a lexicographical order and assigned NEW ranks.***

1. ***Next Iteration***
2. ***Replace sequence ranks from Iteration 1 with sequence ranks from Iteration 2. Order lexicographically and re-rank. Iterations stop when no new ranks can be assigned.***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |  | ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |  | ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |

******

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |  | ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |  | ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |

|  |  |
| --- | --- |
|  |  |
|  |  |

******

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |  | ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |

******

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |  |  | ***A*** | ***A*** | ***X*** | ***A*** | ***A*** | ***A*** | ***A*** | ***X*** |  |  |

******

***Iteration1: RE-RANKING***

***Seq|its rank|linker:rank of the distal seq : position on this-position on distal;…|rank of next distal seq…***

***AAXAAAAX|1|SS:1:3-3;8-8 AAXAAAAX|1|SS:1:3-3|1:8-8 AAXAAAAX|1|SS:1:3-3|1:8-8***

***AAXAAAAX|1|SS:1:3-3;8-8 AAXAAAAX|1|SS:1:3-3|1:8-8 AAXAAAAX|1|SS:1:3-3|1:8-8***

***AAXAAAAX|1|SS:1:3-3|1:8-8 AAXAAAAX|1|SS:1:3-8|1:8-8***

***AAXAAAAX|1|SS:1:3-3|1:8-3 AAXAAAAX|1|SS:1:3-8|1:8-3***

***Iteration2: RE-RANKING***

***AAXAAAAX|1|SS:1:3-3;8-8 AAXAAAAX|3|SS:3:3-3|3:8-8 AAXAAAAX|3|SS:3:3-3|3:8-8***

***AAXAAAAX|1|SS:1:3-3;8-8 AAXAAAAX|3|SS:3:3-3|3:8-8 AAXAAAAX|3|SS:3:3-3|3:8-8***

***AAXAAAAX|3|SS:2:3-3|5:8-8 AAXAAAAX|5|SS:4:3-8|3:8-8***

***AAXAAAAX|2|SS:3:3-3|4:8-3 AAXAAAAX|4|SS:2:3-8|5:8-3***

***Iteration3: RE-RANKING***

***AAXAAAAX|1|SS:1:3-3;8-8 AAXAAAAX|4|SS:4:3-3|4:8-8 AAXAAAAX|4|SS:4:3-3|4:8-8***

***AAXAAAAX|1|SS:1:3-3;8-8 AAXAAAAX|4|SS:4:3-3|4:8-8 AAXAAAAX|4|SS:4:3-3|4:8-8***

***AAXAAAAX|3|SS:2:3-3|6:8-8 AAXAAAAX|6|SS:5:3-8|3:8-8***

***AAXAAAAX|2|SS:3:3-3|5:8-3 AAXAAAAX|5|SS:2:3-8|6:8-3***

***Mixture Hashcode***

Mixtures hashcode is a combination of mixture moieties’ UIDs. Defined in Mixture.cs as:

public override string UID

{

get {

return MixtureSubstance?.Moieties?.UID();

}

}

Where Moieties.UID is defined as (in Moieties.cs) as a MD5 hashed “|” separated string of combined ordered individual moieties:

public static string UID(this IEnumerable<Moiety> mlist)

{

return

mlist.Count() == 0 ?

null :

String.Join("|", mlist.OrderBy(m => m.UID).Select(m => m.UID))

?.GetMD5String();

}

With each individual moiety UID defined as (in Moieties.cs):

Moiety.UID:

public string UID {

get {

return (InChI + Stereo + Amount.UID + (Submoieties.Count == 0 ? "" : "[" + String.Join("|", Submoieties.OrderBy(m => m.UID).Select(m => m.UID)) + "]"));

}

}

Where:

**InChI** – Moiety molecule InChI, defined by Molecule.InChI where Molecule is an instance of a ChemSpider class.

**Stereo** is defined as (in Moieties.cs):

public string Stereo {

get { return SpecialStereo == null ? null : decodeStereoType(SpecialStereo); }

}

decodeStereoType converts SpecialStereo to stereo codes as follows (as defined in Moieties.cs in decodeStereoType(string stereo)):

switch ( stereo.Trim().ToUpper() ) {

case "(+)":

return "C103202";

case "(-)":

return "C103203";

case "(+/-)":

return "C103204";

case "SQUARE PLANAR 1":

case "SP-4-1":

return "C103211";

case "SQUARE PLANAR 2":

case "SP-4-2":

return "C103212";

case "SQUARE PLANAR 3":

case "SP-4-3":

return "C103213";

case "SQUARE PLANAR 4":

case "SP-4-4":

return "C103214";

case "TETRAHEDRAL":

case "T-4":

return "C103215";

case "OCTAHEDRAL 12":

case "OC-6-12":

case "TRANS-OCTAHEDRAL":

return "C103216";

case "OCTAHEDRAL 22":

case "OC-6-22":

case "MER-OCTAHEDRAL":

return "C103217";

case "OCTAHEDRAL 21":

case "OC-6-21":

case "CIS-OCTAHEDRAL":

case "FAC-OCTAHEDRAL":

return "C103218";

case "CAHN-INGOLD-PRELOG PRIORITY SYSTEM":

case "CIP SYSTEM":

return "C103219";

case "AXIAL R":

case "RA":

case "P":

case "DELTA":

case "AXIAL, R":

return "C103220";

case "AXIAL S":

case "SA":

case "M":

case "LAMBDA":

case "AXIAL, S":

return "C103221";

default:

return null;

**Amount.UID** (see Protein hashcode section for explanation of Amounts)

**Submieties**

(Submoieties.Count == 0 ? "" : "[" + String.Join("|", Submoieties.OrderBy(m => m.UID).Select(m => m.UID))

Appends UIDs of all the submoieties of the given moiety separated by “|”. Submoieties are also instances of a Moiety class and their UIDs follow the same logic.

***Chemicals (mol2spl) Hashcode***

Chemicals are represented as moieties and similarly to Mixtures use Moiety hashcodes (see Mixtures section).

***Structurally Diverse Hashcode***

public override string UID

{

get {

StringBuilder sb = new StringBuilder(BibReference);

XElement xCharacteristic = InfraspecificSpl;

if ( xCharacteristic != null )

sb.Append(xCharacteristic.Value);

return sb.ToString().GetMD5String();

}

}

Where:

***BibReference*** is the bibliographic reference referencing the substance.

***IntraspecificSpl*** is an entire SPL Characteristic element with the value of the element set to SourceMaterial.Organism.IntraspecificDescription and ID mapped as follows:

If SourceMaterial.Organism.IntraspecificType is one of "SEROTYPE", "SEROVAR", "SEROGROUP"

Then ID of the SPL Characteristic is set to “serotype”

If SourceMaterial.Organism.IntraspecificType is "CULTIVAR"

Then ID of the SPL Characteristic is set to “cultivar”

If SourceMaterial.Organism.IntraspecificType is one of "STRAIN", "SUBSTRAIN"

Then ID of the SPL Characteristic is set to “strain”

The code defining the above logic is in StructurallyDiverse.cs:

private XElement InfraspecificSpl

{

get {

XElement xCharacteristic = null;

if ( SourceMaterial != null &&

SourceMaterial.Organism != null &&

!String.IsNullOrEmpty(SourceMaterial.Organism.IntraspecificType) &&

!String.IsNullOrEmpty(SourceMaterial.Organism.IntraspecificDescription) )

{

if ( new List<string> { "SEROTYPE", "SEROVAR", "SEROGROUP" }.Contains(SourceMaterial.Organism.IntraspecificType.ToUpper()) )

xCharacteristic = new SplCharacteristic("serotype", SourceMaterial.Organism.IntraspecificDescription).SPL;

else if ( new List<string> { "CULTIVAR" }.Contains(SourceMaterial.Organism.IntraspecificType.ToUpper()) )

xCharacteristic = new SplCharacteristic("cultivar", SourceMaterial.Organism.IntraspecificDescription).SPL;

else if ( new List<string> { "STRAIN", "SUBSTRAIN" }.Contains(SourceMaterial.Organism.IntraspecificType.ToUpper()) )

xCharacteristic = new SplCharacteristic("strain", SourceMaterial.Organism.IntraspecificDescription).SPL;

}

return xCharacteristic;

}

}