Decoding rules:

At both end of a DNA strand, there is primer and backward primer that are sequences representing the element type and showing where it begins and ends. The "Required" part follows the primer, which contains every attribute that is necessary for this element. For every SVG tag, the required attributes are organized and stored in a fixed order, in the DNA sequence that represents it. You can check it in this document. Example is given in Figure 1.

Primer	x: 4bps	y: 8bps	z: xbps	$\cdots$	Backward Primer

Figure 1

the following is the "Optional" part, which contains other supported attributes that are not considered necessary to build this element, but still useful enough. At the beginning there is an unsigned integer implying the total number of attributes listed in this part. Each optional attribute contains information including a key to identify it, the number of nt's to store its value, and its value. Example is given in Figure 2.

total	$\cdots$	key	len	value	$\cdots$

Figure 2

Value Representation:

the DNA sequence representing a certain value begins with a type code, indicating the value type. The following is a stream of nucleotides, with fixed or volatile length. There are 4 types of nucleotide, mapped to the 4 digits of quaternary:

nucleotide	quaternary digit	binary digits
A	0	00
T	1	01
C	2	10
G	3	11

The value representation is based on this mapping principle.

signed integer:
Type code: A(non-negative), T(negative)
Length: volatile
Representation:

A integer's type code may be `A` or `T`. `A` stands for non-nagative numbers and `T` stands for negative ones. A 2-nt sequence is attached to the type code, encoding the length of the quaternary representation of $|x|$ which is stored in the following sequence.

When a integer is not used for representing the value of an attribute, but the size information of some container(e.g., the length of a string or the total number of optional attributes in a tag), the type code is omitted.

A/T	length: 2 nt's	value: *length* nt's

Figure 1. DNA pattern of a signed integer.

floating-point number:
Type code: C
Length: 16 nts
Representation:

We store a floating-point number according to IEEE floating-point standard, with a bit sequence which will be converted into DNA sequence. The form of representation is $V = (-1)^s \times M \times 2^E$. The length of the bit sequence is 32, including a sign bit, a 8-bit exponent field and a 23-bit fraction field.(for details see IEEE 754)

C	value: 16 nt's

Figure 2. DNA pattern of a floating-point number.

string:
Type code: G
Length: volatile
Representation:

A string begins with its length(an integer indicating the number of characters in this string, and as is mentioned before, type code is omitted). We use ASCII code to encode a character. Each character in a string takes up 4 nt's (or 1 byte).

G	length	content: 4**length* nt's

Figure 3. DNA pattern of a string.

Addressing:

In DNA storage, each strand in the DNA pool is a data block. To identify the position of a data block, an address sequence is attached to the strand, so that the information of the whole input string can be restored correctly.

In DNA-SVG encoding, before splitting or merging the strands for appropriate sequence length, a single strand stores one tag(or element) in `.svg` file.

The structure of a `.xml` file can be parsed as a tree. Each tag is a node on the tree. Thus, to identify the position of a tag in `.xml` file means to locate it on the xml tree. In tree storage, each node saves the identifier(a unique address or name) of all of its children. For shorter sequence length, we convert the xml tree to a **left-child, right-sibling binary tree** first. After this modification, the address sequence of a tag should have the following form:

self identifier	left-child identifier	right-sibling identifier

Figure 1. Address sequence.

where we use the **dfs order** of the tag as its identifier.

Merging/Splitting:

We can get a encoded DNA list from the work we have done,but there is a problem that the actual DNA length of strands is negated. So we split long strands and merge short strands for addressing this issue.

Suppose that the Min Length is 150 and Max length is 300. Then 1nt is used to record the type of strands. A for normal, T for long, C for short.

Split Long Strands: split the long stands to several strands. If the the last one is short, then make the last two strands half. *Structure: T + (the id of the original strands) + (the order of current processed strands) + content.*

Merge Short Strands: merge the short ones *Structure : C + (the encoded number of original strands length + content) for each short strands.*

Supported Tags:

<rect>

Required Attributes:

order	name	type
1	x	number
2	y	number

Optional Attributes:

id, class, style, fill, transform

<circle>

Required Attributes:

order	name	type
1	x	number
2	y	number

Optional Attributes:

id, class, style, fill, transform

<g>

Optional Attributes:

id,class,style

<path>

Required Attributes:

order	name	type
1	d	string

Optional Attributes:

id, class, style, fill, transform

<polygon>

Required Attributes:

order	name	type
1	points	number

Optional Attributes:

id, class, style, fill, transform

<style>

Required Attributes:

order	name	type
1	text	str

Optional Attributes:

id, class, style, fill, transform

<ellipse>

Required Attributes:

order	name	type
1	rx	number
2	ry	number
3	cx	number
4	cy	number

Optional Attributes:

id, class, style

<defs>

Optional Attributes:

id, class, style

<title>

Required Attributes:

order	name	type
1	text	string

Optional Attributes:

id, class, style

<filter>

Optional Attributes:

id, class, style, filterRes, filterUnits, fill, transform

<linearGradient>

Required Attributes:

order	name	type
1	x1	number
2	y1	number
3	x2	number
4	y2	number

Optional Attributes:

id, class, style, gradientUnits, gradientTransform, spreadMethod, xlink:href, fill, transform

<stop>

Required Attributes:

order	name	type
1	offset	number

Optional Attributes:

id, class, style, stop-color, stop-opacity, fill, transform

<feOffset>

Required Attributes:

order	name	type
1	in	number
2	dx	number
3	dy	number

Optional Attributes:

id, class, style, result, fill, transform

<feComposite>

Required Attributes:

order	name	type
1	in	str
2	in2	str
3	operator	str

Optional Attributes:

id, class, style, k1, k2, k3, k4, result, fill, transform

<feColorMatrix>

Required Attributes:

order	name	type
1	in	str
2	values	number

Optional Attributes:

id, class, style, type, result, fill, transform

<feMerge>

Optional Attributes:

id, class, style

<feMergeNode>

Required Attributes:

order	name	type
1	in	str

Optional Attributes:

id, class, style

<feGaussianBlur>

Required Attributes:

order	name	type
1	in	str
2	stdDeviation	number

Optional Attributes:

id, class, style, fill, transform edgeNode, result

<line>

Required Attributes:

order	name	type
1	x1	number
2	x2	number
3	y1	number
4	y2	number

Optional Attributes:

id, class, style, fill, transform

<radialGradient>

Required Attributes:

order	name	type
1	cx	number
2	cy	number
3	fr	number
4	fy	number

Optional Attributes:

id, class, style, fill, transform, gradientUnits, gradientTransform,spreadMethod

<use>

Required Attributes:

order	name	type
1	x	number
2	y	number

Optional Attributes:

id, class, style, fill, transform, href

<clipPath>

Optional Attributes:

id, class, style, fill, transform