SimPose-ium file formats

Notation:

All the SimPose-ium specific files are text files.

The description of these files is in terms of combinations of 'tokens', which may be individual characters, keywords or primitives types.

In the following descriptions:

- names in < >'s represent primitive types (see below) or previously defined tokens made up of combinations of other tokens. To make the descriptions more meaningful, the token or type name may optionally be followed by a label - to indicate the purpose of the value. E.g. <int-NumberOfPoses> is an integer that is used to represent the number of poses in a pose library.
- *<int-n> means repeat the previous token 'n' time. The <int> may be explicit (e.g. *10) or may be a label (e.g. *NumberOfPoses)
- *(<int-m>..<int-n>) means repeat the previous token between 'm' and 'n' times. This occurs where the file may contain an arbitary number of similar tokens (usually all starting with the same character or string), the list being terminated by the first token of a different type.
- \n is a significant newline character
- with the above exceptions, all other characters appear, as written, in the files

Tokens are defined as:

<NewToken> = … some combination of previously defined tokens …

Primitive types:

<int>	an integer (may be signed)
<float>	a floating point value - usually to 6 decimal places
<string>	all characters to the end of the current line are treated as a string. This is usually used for names, which can include spaces. No provision is made for control characters or escape sequences, i.e. \t is those two characters not a tab. Note that when SimPose-ium reads a string, it also reads the newline character from the file, but this is not stored as part of the string

A pose

This isn't actually a primitive type, but is used in several of the file formats. Its description involves two additional new tokens:

<vector> = **|** <float-X> <float-Y> <float-Z> **|** \n

<quaternion> = **|** <float-X> <float-Y> <float-Z> <float-S> **|** \n

That is a <vector> is three floats on a line, that starts and ends with '|'. The three floats represent the X, Y and Z components of the vector, in some appropriate units.

A <quaternion> is similar, but with four floats. Quaternions are always normalised, so (x*x + y*y + z*z + s*s = 1.0)

<pose> = **Time** <int-time>
<vector> *2
<quaternion> *29

The Time value has different interpretations in different files, normally (as in a KFD file) it is the time in an animation when the pose applies.

The two vectors are the translation of the ROOT and PELVIS in the pose (in that order)

The 29 quaternions define the rotations for the 29 joints in the order defined in the adult skeleton file.

The Pose Library (*.PSL)

There are actually two versions of the pose library:

- those created by SimPose-ium V4 and earlier, and
- those from V5.0 and later.

V5.0 and later can read the earlier format, but V4 and earlier cannot read the new format.

Both use:

```
<namedpose>  =        <string-PoseName>
                      <pose>
```

The V4 and earlier format is:

```
POSELIBRARY <int-NumberOfPoses> \n
<namedpose> *NumberOfPoses
```

In this format, the 'time' value in each pose is ignored.

The V5 and later format is:

```
POSELIBRARY2   <int-NumberOfPoses >   <int-NumSubStr> \n
<namedpose> *NumberOfPoses
<string> *NumSubStr
```

For each named pose, the 'time' value is interpreted as:

0:	pose applies to the adult skeleton only
1:	pose applies to the child skeleton only
-1:	pose can be applied to either skeleton - means the pose was imported from a library in the old format

NumSubStr is twice the number of SubSets defined in the library. That is, each SubSet is defined by two strings. The first is its name, the second consists of 32 characters.

The first character is '0' or '1':

'0':	SubSet applies to the adult skeleton only
'1':	SubSet applies to the child skeleton only

The other 31 characters refer to possible joint translations and rotations, in the order ROOT translation, PELVIS translation, and 29 joint rotations in the same order as the pose quaternions. The translation/rotation is either required '1' or can be ignored '0'.

The Keyframe File Format (*.KFD)

The Keyframe file format is:

```
KeyFrame <int-NumberOfKeys> <int-Frames> <int-SequenceLength> \n
s <string- StartPoseName>
e <string- EndPoseName>
<pose> *NumberofKeys
```

SequenceLength is the number of animation frames in a complete execution of the animation. That is, the length of the animation in seconds times 30.

Frames is the number of animation frames defined by the key frame poses. For a 'translation' animation, this is the same as SequenceLength, but for a 'bounce' it if half SequenceLength (SequenceLength is always even).

NumberOfKeys is the number of KeyFrame poses defined in the file.

StartPoseName: If this name exists (i.e. the second line may be just "s" on its own), it means that the KeyFrame animation was defined to start with a specific pose selected from a Pose library. When the KeyFrame animation is reloaded, if SimPose-ium has a Pose library loaded that contains a pose of that name, the pose at time=0 will be taken from the library, rather than the data in the KFD file.

EndPoseName: If this name exists (i.e. the third line may be just "e" on its own), it means that the KeyFrame animation was defined to end with a specific pose selected from a Pose library. When the KeyFrame animation is reloaded, if SimPose-ium has a Pose library loaded that contains a pose of of that name, the pose at time=Frames-1 will be taken from the library, rather than the data in the KFD file.

The reason for the above is that the poses at time=0 and time=Frames-1 provide the links from one animation to another. Suppose a change is made to one of these 'link poses' in the pose library, rather than having to change all the keyframe animations that use that pose, this mechanism ensures that the latest version of the link pose is always used.

The poses represent the pose of the Sim at various times during the animation (indicated by the pose's time field). The first pose is always time=0, and the last time=Frames-1. The other poses are ordered by time.

```
Note that:    NumberOfKeys <= Frames
              SequenceLength = Frames    or    SequenceLength = Frames*2
              TimeOfPose[n-1]  < TimeOfPose[n]  < TimeOfPose[n+1]
```

The Animation Sequence File Format (*.SPS)

This file requires information about the Sims being displayed:

```
<object>    =    m <int-PartId> <string-MeshName>
                 t  <int-PartId> <string-TextureName>
```

PartId	Part index, currently 0 to 7. Note the mesh and texture should have the same index. The parts are head, body, hands(*2) + up to 4 props
MeshName	Name of the mesh file (.skn) that defines this object
TextureName	Name of the texture file (.bmp) applied to this object

```
<figure>    =    f  <int-Id> <int-Type> <int-Hidden> <int-Offset> <int-Skel> \n
                 <object-Part> *(4..8)
                 f - end \n
```

Id	An index, currently 0 for Sim1 and 1 for Sim2
Type	Age/Gender, 0=male, 1=female, 2=boy, 3=girl
Hidden	0=Visible, 1=Hidden
Offset	When only one Sim is displayed or there is "no offset" selected, Offset=1 When two Sims are displayed with offset, one has Offset=0, the other =2
Skel	The skeleton index, 0=adult 1=child
Part	For each part of the Sim, define the mesh and texture required. Sims should have at least 4 parts: head, body, left hand and right hand, and may have upto four props attached.

The file also requires information about the nodes in the sequence 'trees'

```
<animation>  =   A <string-AnimName>

<node>       =   N <int-Typ> <int-Par> <int-Nxt> <int-Id> <int-M> <int-N>  \n
                 <animation>*(0..2)
```

AnimName	The name of an animation associated with this node (see table below)
Typ	The type of this node (see table)
Par	The Id of the parent of this node. 0 = no parent, i.e. the root node
Nxt	The Id of the sibling of this node - i.e. the next node to be performed. 0 = no sibling, i.e. the last node in a sequence
Id	A unique identifier for this node, 1+
M	Parameter used for various purposes (see below)
N	Parameter used for various purposes (see below)

The file format is:

```
SimPoseSPS 1 \n
|Start \n
<node-Sim1>*(1..?)
|End \n
|Start \n
<node-Sim2>*(1..?)
|End \n
|Start \n
<node-BothSim>*(1..?)
|End \n
Selected  <int-SelectedSeq>  \n
<figure-Sim1>
<figure-Sim2>
```

	Typ	M	N	Animations
Unset	-1	not used	not used	none
Root	0	not used	not used	none
Loop *n	1	not used	N	none
Loop *m..n	2	M	N	none
Random	3	not used	not used	none
Action	4	Animation for Sim1	Animation for Sim2	0, 1 or 2, depending on how many of M and N are none-zero
Wait	5	not used	not used	none

M/N values for Action:	-2 if the indicated Sim is not being displayed
	-1 if the animation being edited is to be used
	0+ if an animation file (kfd or cfp) is to be used

The order of writing nodes is such that if the sequence tree looks like:-

```
A-----B-----C
|     |     |
|     |     D
|     |     |
|     |     H
|     |
|     E
|
F----G
```

Note that there is an invisible root node, R, that is the parent of A and F.

When a node is written, the siblings of the node are written before that node's data (e.g. because A needs to identify F as its sibling, F has to be written before A). Similarly, the node's data is written before that of its children, so the children can refer to it as their parent.

So the above diagram is written as follows (considering Par, Nxt and Id only) - the rest of the node data in line would refer to the node in **bold**

Par	Nxt	Id	Comment
0	0	**R**	The root has no siblings or parent
R	0	**F**	After the root, A needs to be written, but as A has a sibling, the sibling must be written first. Note that all children point back to the same parent
F	0	**G**	
R	F	**A**	
A	0	**E**	
A	E	**B**	
B	0	**H**	
B	H	**D**	
B	D	**C**	