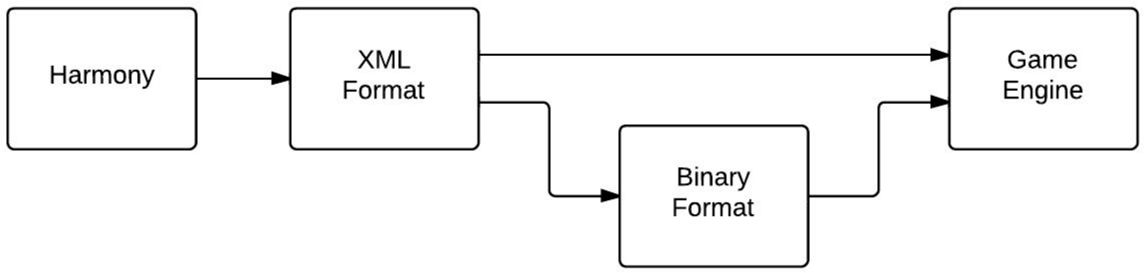
Toon Boom Game SDK
XML Format Documentation

Using the Harmony game export in Harmony allows the integration of animation assets into several known game engines. The Toon Boom Game SDK currently supports Cocos2d-x and Unity through plugins that handle asset management and rendering. The XML data structure that is exported from Harmony can be converted to an optimized binary structure that is interpreted by the plugin running in the chosen game engine.



Game engine pipeline

The XML format is used as a transitory structure to convey animation data to the game engine and can easily be integrated in a wide range of engines. This document will describe in depth the XML data structure so that a new user is able to implement its own integration in a new game engine.

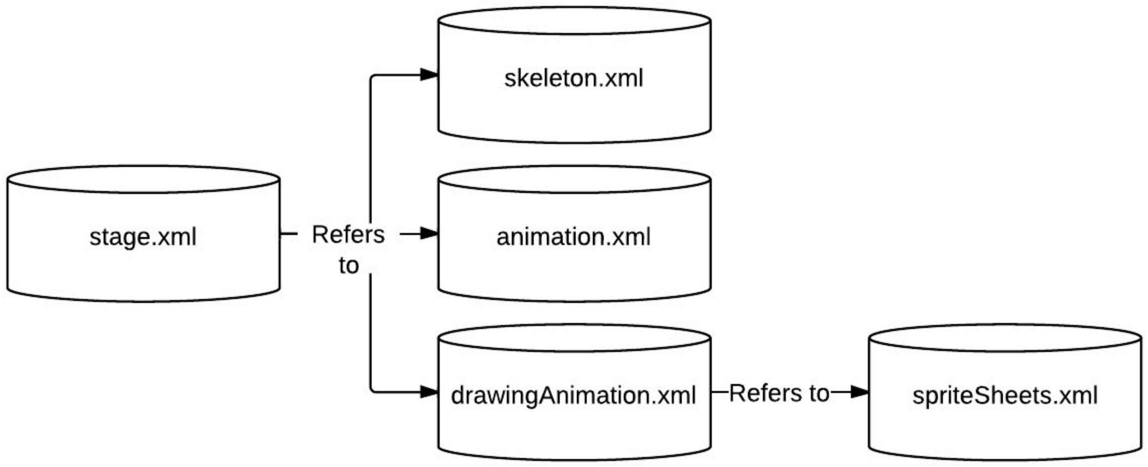
For more information on using Harmony to produce game assets and how to export to the XML data structure, please refer to the appropriate documentation.

Export Format Overview

The exported data structure contains the following files:

- stage.xml
- skeleton.xml
- animation.xml
- drawingAnimation.xml
- spriteSheets.xml

The stage.xml file is the main entry point. For every version of the Harmony scene that is exported to the XML project, a new stage clip entry is created and schedules play sequences that combine a skeleton hierarchy (skeleton.xml), its matching animation (animation.xml) and drawing animation (drawingAnimation.xml).



XML database interrelation

Stage.xml

The stage data structure lists the available stage clips (tag *stage*) that can be played for a single character. The game developer will need to retrieve the named entry manually or build a collection of playable stage clips. The *skeleton* is usually reused from clip to clip, while the *animation* and *drawingAnimation* will change.

A sound event can be triggered in a stage clip (tag *sound*). The sound files are copied in the audio folder and are referenced by the tag *name*. The sound is scheduled to play at a fixed time (tag *time*) after the clip started playing.

Additionally, metadata can be added to a stage clip. Metadata in the Toon Boom Game SDK are user defined values either set for the current scene (stage clip) or a specific node in the scene. They are identified with the tag *meta* and can be retrieved to add custom behaviours in a game. The anchor (tag *anchor*) is a special metadata known in the existing implementations for the Toon Boom Game SDK. The anchor is used to extract a named node in the skeleton in the game to either extract its position or attach objects to it.

Finally, the prop clip (tag *props*) and the props (tag *prop*) are optional objects that are not necessary to the main animation loops, but that can be added to the animation to supplement it (*e.g.* clothes, weapons, items). An anchor node can be used as a reference for the prop to hook itself unto.

```
<stages>
 <stage name="SpaceDuckRun">
  <play name="playDuck" drawingAnimation="runDuck" skeleton="duck" animation="runDuck"/>
  <anchor play="playDuck" node="AnchorLHand"/>
  <meta name="myParameter1" value="myValue1"/>
  <meta name="myParameter2" value="myValue2" node=”RHand”/>
 </stage>
 <stage name="SpaceDuckIdle">
  <play name="playDuck" drawingAnimation="idleDuck" skeleton="duck" animation="idleDuck"/>
  <sound name="sound_effect.wav" time="10"/>
 </stage>
 <props name="SpaceDuckProps">
  <prop name="object" drawingAnimation="objAnim" skeleton=”object" animation="objAnim"/>
 </props>
</stages>
```

Snapshot from stage.xml

Skeleton.xml

To allow an efficient traversal and rendering, the skeleton is exported as a tree data structure. For every skeleton node, there can only be a single parent node, and an ordered list of children nodes. The order of the nodes in the tree are used in conjunction with the position on the z axis to calculate the rendering order of every skeleton node. This order will change through time as a node may move on the z axis.

The skeleton data structure is separated in two parts. The first part defines a flat list of skeleton nodes, either *read* or *peg*, while the second part lists the links between those nodes. The order in which links are listed in the XML file defines the order of the nodes in the skeleton tree.

```
<skeletons>
 <skeleton name="duck">
  <nodes>
   <peg id="0" name="duck_PEG"/>
   <peg id="1" name="duck_UPPER_body_p"/>
   <read id="2" name="duck_FULL_arm_01"/>
   (...)
  </nodes>
  <links>
   <link in="duck" out="0"/>
   <link in="0" out="1"/>
   <link in="1" out="2"/>
```

```
    (...)
   </links>
  </skeleton>
 </skeletons>
```

Snapshot from skeleton.xml

Both the *peg* and the *read* node can be animated, but only the *read* node will refer to a renderable animated drawing.

Additionally, some effects created in the Toon Boom Harmony suite can be exported in the XML data structure. Those are the bone and game-bone deformation hierarchies and the cutter and inverse cutter matte effects.

```
<skeletons>
 <skeleton name="effects">
  <nodes>
   <cutter id="0" name="cutterEffect"/>
   <matte id="1" name="cutterMatte"/>
   <read id="2" name="matteDrawing"/>
   <read id="3" name="cutDrawing"/>
   <deform id="4" name="deformEffect"/>
   <deformRoot id="5" name="deformRoot"/>
   <bone id="6" name="bone1"/>
   <bone id="7" name="bone2"/>
   <read id="8" name="deformedDrawing"/>
  </nodes>
  <links>
   <link in="effects" out="0"/>
   <link in="0" out="1"/>
   <link in="1" out="2"/>
   <link in="0" out="3"/>
   <link in="effects" out="4"/>
   <link in="4" out="5"/>
   <link in="5" out="6"/>
   <link in="6" out="7"/>
   <link in="4" out="8"/>
  </links>
 </skeleton>
</skeletons>
```

Snapshot from skeleton.xml

Both the deformation hierarchy and the cutter effect module as they are shown in Toon Boom Harmony will be modified to fit a tree structure in the XML Data structure. A simple deformation hierarchy containing two bones could be represented like this in a tree structure:

```
deformEffect (deform)
 ↳  deformRoot (deformRoot)
     ↳  bone1 (bone)
         ↳  bone2 (bone)
 ↳  deformedDrawing (read)
```

In this example, the node *deformedDrawing* is child to *deformEffect* and thus will be deformed by the skeleton defined underneath the *deformRoot* node. A similar process is applied to the cutter effects to separate the matte drawing from the drawing being cut. The cutter can either be a positive cutter (tag *cutter*) or negative cutter (tag *cutterInverse*).

Animation.xml

The animation data structure specifies the animation curves used to calculate the transformation matrix with which a skeleton node will be animated. The data structure is separated in two parts. The first part (*attrlinks*) defines a list of attributes and the function they use, while the second part (*timedvalues*) enumerates the functions themselves.

```
<animations>
 <animation name="Run_duck">
  <attrlinks>
   (...)
  </attrlinks>
  <timedvalues>
   (...)
  </timedvalues>
 </animation>
 (...)
</animations>
```

Snapshot from animation.xml

The *attrlink* tag defines an attribute for a named node in the skeleton hierarchy. The link may either be to an animated function (using the *timedvalue* tag) or to a constant value (using the *value* tag). Only the single value attributes can currently export their constant value directly in the *attrlink* tag.

```
<attrlink attr="rotation.anglez" node="duck_PEG" timedvalue="ATV-06D0CEF4C7E1AE2D"/>
<attrlink attr="position.x" node="duck_PEG" timedvalue="ATV-06D0CEF4C7E1AE23"/>
```

```
<attrlink attr="position.y" node="duck_PEG" timedvalue="ATV-06D0CEF4C7E1AE25"/>
<attrlink attr="position.z" node="duck_PEG" timedvalue="ATV-06D0CEF4C7E1AE27"/>
<attrlink attr="scale.x" node="duck_PEG" value="1.0"/>
<attrlink attr="scale.y" node="duck_PEG" value="1.0"/>
<attrlink attr="skew" node="duck_PEG" timedvalue="ATV-06D0CEF4C7E1AE2F"/>
<attrlink attr="pivot" node="duck_PEG" timedvalue="PIVOT-TV-duck_PEG"/>
```

Sample attribute link in animation.xml

Attribute Name	Values	Description
position.attr3dpath or offset.attr3dpath	3	Defines the position of the node on the xyz axis. This attribute can only use a catmull function as its animation curve. If this attribute is set, then no separate position attributes can be used. The z value of the path will be used to sort layers.
position.x or offset.x	1	Defines the position of the node on the x axis.
position.y or offset.y	1	Defines the position of the node on the y axis.
position.z or offset.z	1	Defines the position of the node on the z axis. This value will also be used to sort layers.
scale.x	1	Defines the scaling of the node on the x axis.
scale.y	1	Defines the scaling of the node on the y axis.
scale.xy	1	Defines the uniform scaling of the node. If this attribute is set, then no separate scale attributes can be used.
rotation.anglez	1	Defines the rotation of the node on the z axis.
skew	1	Defines the vertical skewing of the node.
velocity	1	Defines the velocity curve used by the 3d path if set.
pivot	2	Defines the pivot of the node on the xy axis.

List of valid attributes for a single node

Harmony currently exports four possible functions to the XML data structure. Those are:

- The Bezier curve for single value attributes.
- The linear curve for single value attributes.
- The Catmull curve for attributes with three values.
- The pivot curve for the pivot attribute.

The Bezier curve in Harmony is a standard cubic Bezier curve. The x axis of the curve represents the frame number, while the y axis represents the value. Every point in the curve defines a 2D position referenced by tags (*x*, *y*) and a left and right handle respectively referenced by (*lx*, *ly*) and (*rx*, *ry*). The tag *constSeg*, when set to true, will maintain a constant value in the animation until the next point.

```xml
<bezier name="ATV-06852082F5FDF142">
 <pt x="1" y="-778.482" rx="1" ry="-778.482" lx="1" ly="-778.482" constSeg="false"/>
 <pt x="30" y="-35.64" rx="55.68" ry="-47.723" lx="3.6" ly="-41.930" constSeg="false"/>
 <pt x="60" y="677.531" rx="60" ry="677.531" lx="60" ly="677.531" constSeg="false"/>
</bezier>
```

Sample bezier curve in animation.xml

To ensure the calculation of the Bezier curve matches exactly the one made in Harmony, certain guidelines must be followed. As the z value is used as a comparison basis to sort the order layers appear in, an imprecise calculation might lead to display issues afterwards. This is also true for the catmull curve calculation. Please refer to file TV_BezierCurveView.cpp in the Toon Boom Game SDK for implementation details.

If a custom implementation of the cubic Bezier curve is not possible, then, it is important to increase the values between each layers on the z axis to avoid rounding errors in the calculations.

The Catmull curve in Harmony is a modified version of the Kochanek-Bartels cubic spline (or TCB spline) which integrates tension, continuity and bias parameters to further control the shape of the curve.

Compared to the initial algorithm, the curve in Harmony uses non-uniformly scaled coordinates. Values on the y axis are multiplied by 0.75 and values on the z axis are multiplied by 2. Those adjustments to the coordinates must be applied when calculating derivative and position.

Furthermore, the calculation of the first (1) and last derivative (2) of the curve are modified to avoid numerical instabilities.

$$T_0^O = 2(P_1 - P_0) - T_1^I \tag{1}$$

$$T_n^I = 2(P_n - P_{n-1}) - T_{n-1}^O \tag{2}$$

The Catmull curve defines three scaling parameters (*scaleX, scaleY and scaleZ*) that are used to convert the curve coordinates to the exported game in pixel coordinates. Each Catmull point defines a 3D position (*x*,*y*,*z*) and optionally the parameters *tension*, *continuity* and *bias*. A catmull point may be locked in time at a specific frame or a control point that can move around.

The tag *constSeg*, when set to true, will maintain a constant value in the animation until the next point.

```
<catmull name="ATV-06851952CF989EAB" scaleX="60" scaleY="45" scaleZ="0.083333">
 <pt x="-12.71584409540883" y="1.337374612239777" z="0" lockedInTime="1"/>
 <pt x="-0.09544972030904829" y="9.808132841700422" z="0"/>
 <pt x="11.712813136229" y="0.6902578643818211" z="0" lockedInTime="60"/>
</catmull>
```

Sample catmull curve in animation.xml

The velocity function (linked by the attribute *velocity*) must be used with the Catmull curve to calculate the distance in the curve at a specific frame.

The implementation of the Catmull curve is specific to Harmony and as such, it would be difficult to match the exported data to any other existing software implementation. Consequently, if it is not possible to implement a custom calculation for the Catmull curve, it is advised to either use separate position curves or export animation data as linear curves instead.

Please refer to files TV_CatmullCurveView.cpp, TR_CatmullCompute.cpp and TR_NodeTreeBuilder.cpp in the Toon Boom Game SDK for implementation details.

The linear curve can be used if the game requires efficiency, but does not need smooth interpolation between frames. Each linear point defines a 2D position referenced by tags (*x*, *y*).

```
<linear name="ATV-06852082F5FDF142">
 <pt x="1" y="-778.482"/>
 <pt x="2" y="-644.88"/>
 <pt x="3" y="-560.544"/>
 (...)
 <pt x="58" y="494.766"/>
 <pt x="59" y="570.881"/>
 <pt x="60" y="677.531"/>
</linear>
```

Sample linear curve in animation.xml

The pivot curve sets pivot positions at fixed frames. Each pivot set is valid from that frame until it is changed. Each pivot point defines a 3D position referenced by tags (*x*, *y*, *z*) and a starting frame referenced by tag *start*.

```
<pivot name="PIVOT-TV-duck_PEG">
 <pt x="-131.841438685421" y="197.0318509914551" z="0" start="1"/>
 <pt x="-131.7749938750305" y="200.7527608059082" z="0" start="66"/>
 <pt x="-131.841438685421" y="197.0318509914551" z="0" start="99"/>
</pivot>
```

Sample pivot in animation.xml

All the attributes attached to a skeleton node define a transformation matrix for that node (3). The final transformation matrix for a sprite is defined by the chain in the hierarchy of the skeleton.

$$M = M_{pivot} M_{offset} M_{rotate} M_{skew} M_{scale} M_{pivot}^{-1} \tag{3}$$

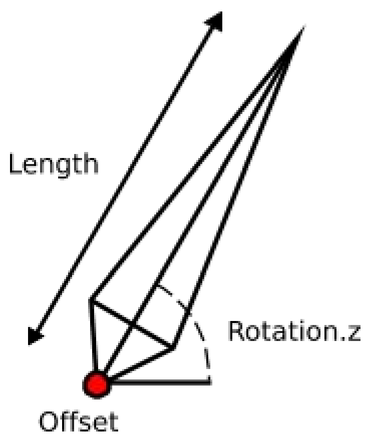
The z value of the offset attribute must be rounded to a fixed quantum in order to match the value used in Harmony and to properly order the layers on the z axis (4).

$$z' = floor(65536\, z + 0.5) / 65536 \tag{4}$$

The parameters set for a bone node define the resting position (*rest.**) and the animated deformed position (*deform.**) of the bone. The offset parameter is the starting position of the bone and is relative to its parent bone or animated node. The length define the distance to the end of the bone and the rotation, its angle on the z axis.

Attribute Name	**Values**	**Description**
rest.offset.x	1	Resting offset of the bone on the x axis.
rest.offset.y	1	Resting offset of the bone on the y axis.
rest.length	1	Resting length of the bone.
rest.rotation.z	1	Resting orientation of the bone.
deform.offset.x	1	Deformed offset of the bone on the x axis.
deform.offset.y	1	Deformed offset of the bone on the y axis.
deform.length	1	Deformed length of the bone.
deform.rotation.z	1	Deformed orientation of the bone.

List of valid attributes for a single bone node



Bone parameters

DrawingAnimation.xml

The drawing animation data structure specifies the drawings that are to be rendered at specific frames for skeleton nodes. Each *drawingAnimation* set specifies the sprite sheet with which it will render (tag *spritesheet*).

Each drawing entry specifies a collection of exposures (tag *drw*) that are to be rendered for a named node in the skeleton hierarchy. Each exposure is set to be rendered at a specific *frame* for a number of frames (tag repeat). It also refers to an entry in the sprite sheet (tag *name*).

```
<drawingAnimations>
 <drawingAnimation spritesheet="sheet_duck" name="Run_duck">
  <drawing node="duck_antenna" name="10">
   <drw frame="1" repeat="4" name="10-antenna-2.png"/>
   <drw frame="5" repeat="8" name="10-antenna-1.png"/>
   <drw frame="13" repeat="7" name="10-antenna-2.png"/>
   <drw frame="20" repeat="4" name="10-antenna-1.png"/>
   <drw frame="24" name="10-antenna-2.png"/>
  </drawing>
  (...)
 </drawingAnimation>
</drawingAnimations>
```

Snapshot from drawingAnimation.xml

SpriteSheets.xml

The sprite sheet data structure lists the location of each sprite in the sprite sheet image. Each sprite sheet has a *resolution* name that can be set during the export. There can be several sprite sheets with the same name, but with different resolutions. The sprite sheet is referenced on disk by its *filename* that is relative to the export directory.

Each sprite has a 2D offset (*offsetX*, *offsetY*) and scale (*scaleX*, *scaleY*) parameters that are used to offset sprite to its center coordinates and adjust its size to the resolution. A rectangle (tag *rect*) is used to delimit the coordinates of the sprite in the sprite sheet.

```xml
<spritesheets>
 <spritesheet resolution="ipad" name="sheet_duck"
filename="spriteSheets/sheet_space_duck-ipad.png">
  <sprite name="10-antenna-1.png" rect="327,172,167,133" scaleX="0.703125" scaleY="0.703125"
offsetX="-202.237" offsetY="210.39"/>
  <sprite name="10-antenna-2.png" rect="877,58,171,93" scaleX="0.703125" scaleY="0.703125"
offsetX="-202.218" offsetY="192.36"/>
  (...)
 </spritesheet>
</spritesheets>
```

Snapshot from spriteSheets.xml