Ninja: XML Data Specifcation Revison 6

This document specifies the format of the XML data that is used to describe all objects, levels and menus in the game.

2008

Microsoft Game Studios

7/13/2008

Contents

[Why XML? 3](#_Toc206310086)

[Format for Level XML 3](#_Toc206310087)

[Collision line file XML 4](#_Toc206310088)

[Animation file XML 5](#_Toc206310089)

[String file XML 6](#_Toc206310090)

# Why XML?

XML was chosen as a format for the game’s data because of its flexibility, readability and ability to easily integrate itself with other tools and technologies- such as our command line console which will be used to edit levels in-game and on the fly.

# Format for Level XML

The format for level xml looks something like this:

<data>

*<PlayerNinja>*

*<PositionX> 0.5 </PositionX>*

*<PositionY> 2 </PositionY>*

*<Health> 200 </Health>*

*</PlayerNinja>*

*</data>*

The meaning of each tag is as follows:

1. <data> specifies the start of the level document.
2. <PlayerNinja> is the C# class name of an object. This specifies the start of an object’s data. The name given must be a valid C# GameObject derived class. <Camera> or other tags can also be used to make different types of objects.
3. Each sub tag under the <PlayerNinja> specifies a variable associated with that object and the value is contained in between the tags.
4. NOTE: Both object and variable names are case sensitive because of the case sensitivity of the programming language. E.G <PlayerNinja> and <playerNinja> are not the same! Similarly so for <Health> and <health>.

The game can read levels stored in this format and construct the objects automatically. If something goes wrong whilst the data is being read or if the data is invalid the object is free to ignore the problem or throw an exception. If an exception is thrown the object will be immediately removed from the game and the game will continue parsing the rest of the level xml file as normal.

# Collision line file XML

We are using XML files to store collision lines for sprites because storing them for each object and imputing them every time would be too tedious and very redundant. The format for a collision lines xml file is very simple:

<data>

<line>

<coord>-80</coord>

<coord>-18</coord>

<coord>-28</coord>

<coord>45</coord>

</line>

</data>

1. The <data> tag specifies the start of the document.
2. <line> designates data for one line
3. <coord> specifies a coordinate in a line. There must be exactly four coordinates in each line or the line will be ignored. The order of the coordinates is as follows:
   1. Point 1 – x
   2. Point1 – y
   3. Point2 – x
   4. Point2 – y

# Animation file XML

Animations will also be stored in xml files for similar reasons to line files. The format for an animation file is as follows:

<data BoxDimensionsX=32 BoxDimensionsY=64 EllipseDimensionsX=32 EllipseDimensionsY=64>

<Legs OffsetX = “0” OffsetY = “0” SizeX=”128” SizeY=”128”>

<Running FrameRate=”15.2” MaxFrameRate=”20” MinFrameRate=”10”>

<frame>gfx\runnng1</frame>

<frame>gfx\runnng2</frame>

<frame>gfx\runnng3</frame>

</Running>

</Legs>

</data>

The meaning of each tag is as follows:

1. <data> specifies the start of the document’s data.
2. The ‘BoxDimensions’ and ‘EllipseDimensions’ attributes pair are used mainly only for character animations and can be omitted if needed. They suggest a bounding box and bounding ellipse size for the character. The bounding ellipse is used for smooth collision detection with the level, whereas the bounding box is used for hit detection with projectiles and swords etc.
3. <Legs> specifies the data for a part of the animation. Any arbitrary name can be used here- such as <Arms> or <Head> etc. The attributes ‘OffsetX’ and ‘OffsetY’ of this tag specify an offset for this part from the center of the object being animated. The in-game object using this animation should use this information to correctly position each part of the animation. ‘SizeX’ and ‘SizeY’ specify how big the part should be drawn in the game.
4. <Running> specifies a sequence for that part. Again, any arbitrary name can be given such as <Jumping> etc.. The ‘FrameRate’ attribute specifies the speed that this animation should play at.  
   ‘MaxFrameRate’ serves as a guide for the software for some sequences that are sped up / slowed down as to the maximum frame rate the sequence may be played at; it may not necessarily be obeyed. ‘MinFrameRate’ serves a similar purpose except it is the minimum desired frame rate. These two fields are mainly used for character animation.
5. Under the sequence, individual frames are stored within the <frame> tag. The name of an actual texture file for that frame is nested in between the start and end of the tag.

# String file XML

Files holding strings for the game are specified as follows:

<data>

<section\_name1 \_locID=”Blah1”>Some text1</section\_name1>

<section\_name2 \_locID=”Blah1”>Some text1\nAnd a new line</section\_name2>

</data>

1. Firstly <data> is the root element as with all other types of XML files used with the game.
2. Strings are stored under sub-tags which can have any name as shown above.
3. The id of the string is stored under the “\_locID” attribute of the tag. If this is not given the string will be ignored. If it is not unique then duplicate instances of the same named string will be ignored.
4. The inner text of the tag is the string itself. Note that new lines will be removed and leading/trailing whitespace will be erased also when the xml file is read. If a new line is required the special escape symbol \n can be used.