Ninja: XML Data Specifcation Revison 1

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

6/16/2008

# 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.

# Basic format of the XML

Each object will begin first with a tag which must correspond to a recognized C# class in the game. The attributes of this object will be stored in sub tags that are nested underneath the class tag:

*<PlayerNinja>*

*<PositionX> 0.5 </PositionX>*

*<PositionY> 2 </PositionY>*

*<Health> 200 </Health>*

*</PlayerNinja>*

How it will work in the game is something like this:

1. The game will read the PlayerNinja tag and determine that the data that follows underneath is for an object of this type. If the object type is not recognized all the data underneath will be ignored and the object will not be made.
2. A filter pass will be applied to isolate XML specific to platform or locale. See the next section for details of how this process will work.
3. The game will then construct a default PlayerNinja object with all it fields and attributes set to their default settings.
4. The game will then pass control to the newly created PlayerNinja object and give it the data contained within the tags PositionX, PositionY etc. and allow the PlayerNinja object to initialize it’s data using these fields.
5. If any error occurs while the PlayerNinja object is reading its data it is free to either ignore the error or complain and raise an exception. If an exception is raised the game will catch this exception and delete the invalidly created object. After this has happened the rest of the XML will continue parsing as normal. If an exception occurs and the game is in a debug build, the PlayerNinja object may print messages to a debug message output stream to indicate the exact problem that occurred.

# Locale specific XML

Locale specific XML data will be supported as part of the XML format used by the game. Before the game allows an object to read its XML data a filter will be applied where appropriate to isolate locale or platform specific XML data:

<MenuImage>

<comment> Default splash text: American English. </comment>

<file>gfx\menu\en-us\splashtext</file>

<comment> Italian splash text </comment>

<file locale=”it-it”>gfx\menu\it-it\splashtext</file>

<comment> German splash text </comment>

<file locale=”de-de”>gfx\menu\de-de\splashtext</file>

<comment> French splash text </comment>

<file locale=”fr-fr”>gfx\menu\fr-fr\splashtext</file>

<comment> French language splash text for the English region </comment>

<file locale=”fr-en”>gfx\menu\fr-en\splashtext</file>

</ MenuImage >

How it will work as follows:

1. The game will look at all the data tags contained within the MenuImage object.
2. If it sees duplicate tags (<file> in this case) it will see which tags a have valid locale attribute like the examples above. The local attribute text is in the format of language-region.
3. After finding tags with locale attributes, the game will first attempt to match one of the locale specific <file> tags to the current language of the machine. If a match is made then the tag which produced the match will be set as the current tag ‘in-use’.
4. If no matching language <file> tag is found it will attempt to revert to a default language version of the tag which has no locale-attribute (the first tag in this case). If this is not found then the first tag encountered will be used instead! For example “it-it” would be used if the first <file> tag were removed in this case.
5. If a successful match to a language specific <file> tag is made, then the game will attempt to further refine its search by attempting to match a region under that language. If the user’s local is “fr-en” in this case (French English- ridiculous example I know!) then the <file> tag with locale set to “fr-en” will used instead of “fr-fr”. If no region match is made (say the users locale is “fr-de”) then the game will simply use the “fr-fr” tag instead.

# Platform specific XML

Platform specific XML will be supported in a similar manner to locale specific xml. A ‘platform’ attribute specified as part of a data tag will activate platform specific filtering and the game will attempt to select the correct tag for the platform the user is running the game on.

Supported platform types are:

* “Windows”
* “XBOX-360”

For example:

<LevelObject>

<texture>gfx\level\objects\light1</texture>

<comment> Default shader used on platforms other than Xbox 360 or windows</comment>

<shader>fx\neutral\lightShader</shader>

<comment> XBOX version of the shader</comment>

<shader platform=”XBOX-360”>fx\xbox\lightShader</shader>

<comment> Windows version of the shader</comment>

<shader platform=”Windows”>fx\windows\lightShader</shader>

</LevelObject>

If the user is running windows then the last <shader> tag will be used. If the XBOX is being used then the second <shader> tag will be selected. If a platform match cannot be made then a default <shader> tag with no platform attribute will be used.

Platform attributes can also be combined with locale attributes if required:

<shader platform=”Windows” locale=”fr-fr”>fx\windows\lightShader</shader>