NINTENDO
NITRO-System

Data File Format Rules

An Explanation of the Data File Format

Version 1.0.0

Contents

Codes

Tables

Figures

Revision History

Version	Revision Date	Description
1.0.0	5/11/2004	Initial Release

1 Introduction

NITRO-System has a minimal set of rules for the formatting of data files that are output by the tools. This document explains these file format rules.

2 XML File Rules

XML is being increasingly utilized for text-format data files in order to facilitate the handling of text-format data. NITRO-System has a set of rules regarding the use of the XML format for text files. The XML-format text files that are supplied with NITRO-System follow these rules.

2.1 XML File Structure

The NITRO-System XML documents have a basic structure that stores one `head` element and one `body` element in the `root` element and the information that should be stored in the `head` element. The information for the body of the document is defined for each file format, and stored in the `body` element.

XML documents defined in NITRO-System contain the elements described below. Except where it is stated, the order of these elements and the attributes inside these elements are not defined. The element tags must be checked and processed in order to interpret the XML document.

2.1.1 XML Declaration

XML files begin with the XML declaration. This declaration ensures that the XML processor correctly processes the written contents as an XML document.

2.1.2 The `root` Element

The `root` element describes the name of the document. The `root` element has one `version` attribute that describes the document's version. Also included inside of the `root` element is one `head` element and one `body` element that are stored in this respective order.

2.1.3 The `head` Element

The `head` element stores information that pertains to the overall document, including the creator, the time of creation, and comments. The head element also stores the following four elements:

Table 2-1 The `head` element

`create`	Information that relates to the creation of the document.
`title`	The document title.
`comment`	Comments about the document.
`generator`	Information that relates to the generator.

2.1.4 The `create` Element

The `create` element stores information regarding the creation of the document. This element has only

the four attributes shown below, and there are no contents.

Table 2-2 The `create` element

user	Creator.
host	Name of machine used for creation.
date	The time of creation, expressed in the format: YYYY-MM-DDThh:mm:ss. This is an ISO8601 format, an expanded format that gives the date and the local time. (Example: 2002-01-17T17:03:00)
source	If there is source data, this attribute represents the string that indicates the data. The *create* element will not have this attribute if the document is newly created.

2.1.5 The `title` Element

The `title` element stores the title of the document (character data).

2.1.6 The `comment` Element

The `comment` element stores comments (character data) about the document. This is an optional element.

2.1.7 The `generator` Element

The `generator` element stores information that relates to the document's generator. The element has only the following two attributes, and there is no content. If the document has been mechanically created by the application, these elements always exist.

Table 2-3 The `generator` element

name	Generator name
version	Generator version

2.1.8 The `body` Element

The `body` element stores information about the overall document. The contents are defined for each data format type.

2.2   Example of an XML File

Here is an example of an XML document defined in NITRO-System.

Code 2-1 Example of an XML file

```
<?xml version="1.0" encoding="Shift-JIS"?>

<sample version="1.0">

  <head>
    <create user="UserName" host="HostName" date="2002-01-18T10:15:00" source="s.xml" />
    <title>NITRO-System XML</title>
    <comment>Example XML document </comment>
    <generator name="ToolName" version="1.0" />
  </head>

  <body>
    <!--  The content here is defined for each data format -->
  </body>

</sample>
```

3   Binary File Rules

NITRO-System also has rules for binary files, which tend to be relatively large. The rules described here relate to the basic parts of the binary file structure. The rules governing the data contents depend on the various binary file formats.

3.1   Binary File Structure

Binary files are divided into two components: a header part and a data part, as shown in the figure below. In the binary file structure, the header part is always stored first. The data part that follows the header part stores data blocks that contain various types of binary data.

Binary file header	Header part
Data block 1	
Data block 2	
Data block 3	Data part
. . .	

Figure 3-1 Binary file structure

3.1.1   Header Part

The header part stores a single binary-file header that is at least 16 bytes in size. This binary file header stores information for identifying the type of the binary file. Information that is unique to each binary file format can be added to this binary file header. The size of this header must be an integral multiple of 4 (4-byte alignment).

3.1.2   Data Part

The data part stores a number of data blocks. An 8-byte data block header sits at the start of each data block. Data blocks are only defined to have the data block header at the start the data block, and the data block contents are defined for each type of binary data format.

The size of each data block must at least be an integral multiple of 4 (4-byte alignment). Depending on the rules for the various binary file formats, the data block size may be aligned in a larger unit. Moreover, the order in which data blocks are stored is defined for each binary file format type. (Whether the data blocks are arranged in unspecified or specified order varies depending on the type of the binary file format.)

3.2　The Binary File Header

The structure of the binary file header is shown in the following table. The binary file header has at least 16 bytes in size. The binary file header stores the signature indicating the binary file type, and its version number, as well as the code for determining the endian method, binary file size, and the number of data blocks contained in the binary file.

Table 3-1　Binary file header

Type	Parameter name	Meaning	Size
char[4]	signature	File signature	4 bytes
u16	byteOrder	Byte-order mark (for determining endian method)	2 bytes
u16	version	Binary file format's version number	2 bytes
u32	fileSize	Size of binary file	4 bytes
u16	headerSize	Size of binary file header	2 bytes
u16	dataBlocks	Number of data blocks	2 bytes

3.2.1　`signature`

`signature` stores the file signature, which is used to determine the binary file type.

This file signature is a 4-byte code that is stored in memory in order from the highest bit, regardless of the endian method. Note that all other data is dependent on the endian method.

This file signature is stored as four ASCII characters and managed to avoid duplication among the different binary file types.

3.2.2　`byteOrder`

`byteOrder` stores the byte order mark (Zero Width No Break Space) 0xfeff that is used to determine the endian method. The byte data string is stored in order of 0xfe,0xff for big-endian, and 0xff,0xfe for little-endian.

Binary files for NITRO are created using the little-endian method. The information is set in consideration of cases where the same file format is shared with game machines that use a different byte order.

3.2.3　`version`

`version` stores the binary file format's version number. The upper byte stores the major version (integral value), while the lower byte stores the minor version (decimal value). Thus, the version number 1.2 would be the value 0x0102.

3.2.4　`fileSize`

`fileSize` stores the total size of the binary file, including the binary file header.

3.2.5 `headerSize`

`headerSize` stores the size of the binary file header. The size of the binary file header is prescribed to be 16 bytes; therefore, a value 16 is normally stored. If the binary file header store information that is unique to the different binary file formats, `headerSize` stores the total size, including this unique information. The size of binary file header must be an integral multiple of 4 (4-byte alignment).

The very first data block that is included in the binary file is positioned away from the start of the binary file (binary header) by the exact number of bytes that is specified in `headerSize`.

3.2.6 `dataBlocks`

`dataBlocks` stores the number of data blocks that are included in the data part of the binary file. The `dataBlocks` value is an unsigned 16-bit value. One binary file can store up to 65,535 data blocks.

3.3   Data Block Header

Each data block in the data part of the binary file starts with a data block header that has the structure shown below. The data block header is 8 bytes in size, and it stores the type and size of the data block.

Table 3-2 Data block header

Type	Parameter name	Meaning	Size
u32	kind	Data block type	4 bytes
u32	size	Data block size	4 bytes

3.3.1 `kind`

`kind` stores a 4-byte code that describes the data block type. The codes are prescribed for each binary file format. In other words, if the binary file signatures are different, the contents of the two blocks will differ even if they both have the same value for `kind`. Normally data block kinds are composed of four ASCII characters.

3.3.2 `size`

`size` stores the size of the data block, including the data block header. The size of the data block must be a value that is a multiple of 4 (4-byte alignment).

If there are more than one data block, each successive data block is positioned away from the preceding data block header by the exact number of bytes that are specified in `size`.