TWL-System G2D Runtime Binary Format

A Description of the File Format

2009/04/01

**The content of this document is highly confidential
and should be handled accordingly.**

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

Tables

Figures

Revision History

Revision Date	Description
2009/04/01	Revised the description of conditions under which the CPOS block is output.
2008/09/12	Updated the NFTR file format.
2008/05/30	Made revisions in line with the name change (updating from NITRO-System to TWL-System). Corrected typographical errors.
2007/03/14	Updated the NFTR file format.
2005/09/01	Corrected typos. Changed the NCGR and NCBR file formats. Added a description. Added a description of user extended attributes.
2005/05/25	Added NFTR file format.
2005/01/31	Added a description of OAM attribute data to cell information. Added warning about character data size information.
2005/01/XX	Changed the multicell data format. Deleted `NNSG2dMultiCellData.numTotalOams`. Added `NNSG2dMultiCellData.numCellAnim`. Changed the contents of `NNSG2dMultiCellHierarchyData.nodeAttr`.
2004/12/06	Standardized phrases.
2004/10/12	Added a description of cell data that has rectangle region information.
2004/10/04	Added a description of palette compression data.
2004/09/16	Described the file identifier information.
2004/09/02	Added the NSCR files format.
2004/09/02	Support for the September 2 version. Standardized notation.
2004/08/02	Support for the August 2 version. Revised for converter name change.
2004/07/20	Reflected the format changes in the July 20 version.
2004/06/22	Reflected format changes in the June 22 version.
2004/05/27	Reflected format changes.
2004/05/11	Initial version.

1 Introduction

The G2D Runtime Binary Format is a binary file format that can be processed by the G2D runtime. This document describes the G2D Runtime binary format.

1.1 G2D Runtime Binary Format

To create a G2D runtime binary, the intermediate binary file output from NITRO-CHARACTER must be converted with the `g2dcvtr.exe` binary converter. See the binary converter documentation for details on `g2dcvtr.exe` (`TwlSystem\docs\G2D\g2dcvtr_Manual.pdf`).

The following sections provide a description of specific file formats.

2 G2D Runtime Binary Format

This chapter describes the G2D runtime binary format.

2.1 G2D Runtime Binary Types

G2D runtime binaries have the following formats according to the type of data that is stored.

Table 2-1 G2D Runtime Binary Types

Extension	Definition	Description	File Identifier
`.ncer`	Nitro CEll for Runtime	Cell definition information	NCER
`.nanr`	Nitro ANimation for Runtime	Animation definition information	NANR
`.nmar`	Nitro Multicell Animation for Runtime	Animation definition information	NMAR
`.ncgr`	Nitro Character Graphics for Runtime	Character data	NCGR
`.ncbr`	Nitro Character Bitmap format for Runtime	Character (bitmap)	NCBR
`.nclr`	Nitro CoLor palette for Runtime	Color palette	NCLR, NCPR (old version)
`.nmcr`	Nitro MultiCell for Runtime	Multi-cell definition information	NMCR
`.nenr`	Nitro ENtity for Runtime	Entity definition	NENR
`.nscr`	Nitro SCreen for Runtime	Screen definition	NSCR

2.2 TWL-System Binary File Protocol

G2D runtime binaries use the file structure that follows the TWL-System binary file protocol.

Accordingly, the format of binary file header + binary blocks (arbitrary number) is used as determined by the TWL-System binary file rule for the general file structure.

For more information, see the documentation on the TWL-System binary file protocol (`TwlSystem\docs\Readme\DataFormatRule.pdf`).

The following describes each file format in detail.

2.3 Offset (Pointer) Notation

The following uses offset (pointer) notation. It records the offset byte value and indicates the variables that are used as pointer members that perform address conversions at run time.

3 NCER (Cell Definition Information)

NCER is a file format that stores cell definition information. It is created by converting an NCE NITRO-CHARACTER binary file. NCER has a general block structure as shown below.

Table 3-1 NCER Block Structure

Data Block Name	Identifier	Value	Condition	Notes
Cell Bank Block	NNS_G2D_BLKSIG_CELLBANK	'CEBK'	Required	Specific
User Extended Information Block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared
Name Label Information	NNS_G2D_BLKSIG_NAMELABEL	'LABL'	Optional	Shared

The NCER file's specific block is described below.

3.1 Cell Bank Block

This block defines the cell information. The general structure of this block is shown below.

Figure 3-1 NNSG2dCellBankBlock Structure
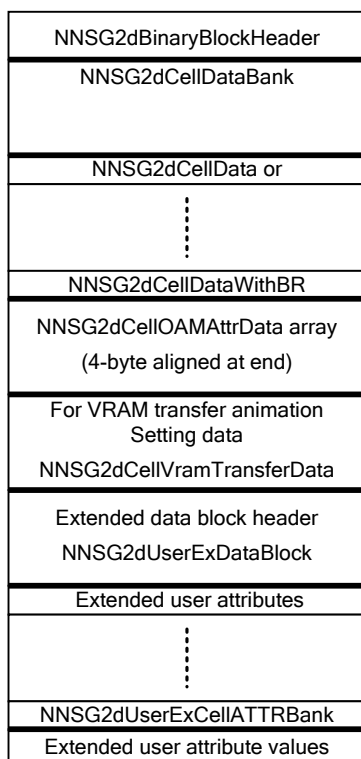
The following tables show the contents for each data structure.

Table 3-2 NNSG2dCellBankBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	8
NNSG2dCellDataBank	cellDataBank	Cell data bank	24

Table 3-3 NNSG2dCellDataBank

Type	Parameter Name	Description	Bytes
u16	numCells	Number of Cell data sets.	2
u16	cellBankAttr	Cell bank attributes.	2
NNSG2dCellData*	pCellDataArrayHead	Offset (pointer) to cell data array.	4
NNSG2dCharacterDataMapingType	mappingMode	Mapping format for reference character.	4
NNSG2dVramTransferData*	pVramTransferData	Offset (Pointer) to data used for VRAM transfer animation (reserved). NULL if not used.	4
Void*	pStringBank	Pointer to string bank (set during execution).	4
Void*	pExtendedData	Pointer to library extended information (unused).	4

Table 3-4 NNSG2dCellData

Type	Parameter Name	Description	Bytes
u16	numOAMAttrs	Cell structure OAM attribute count.	2
u16	cellAttr	Other information on the cell (flip feature usage condition, etc.)	2
NNSG2dCellOAMAttrData*	pOamAttrArray	Offset (pointer) to Cell structure OAM attribute array.	4

Table 3-5 NNSG2dCellBoundingRectS16

Type	Parameter Name	Description	Bytes
s16	maxX	Cell boundary maximum X value	2
s16	maxY	Cell boundary maximum Y value	2

Type	Parameter Name	Description	Bytes
s16	minX	Cell boundary minimum X value	2
s16	minY	Cell boundary minimum Y value	2

Table 3-6 NNSG2dCellDataWithBR

Type	Parameter Name	Description	Bytes
NNSG2dCellData	cellData	Cell data.	8
NNSG2dCellDataWithBR	boundingRect	Rectangle area information.	8

Table 3-7 NNSG2dVramTransferData

Type	Parameter Name	Description	Bytes
U32	szByteMax	Maximum number of bytes being VRAM transferred.	4
NNSG2dCellVramTransferData	pCellTransferDataArray	Offset (pointer) to beginning of the cell transfer information array.	4

Table 3-8 NNSG2dCellVramTransferData

Type	Parameter Name	Description	Bytes
U32	srcDataOffset	Offset value from the beginning of the transfer source data.	4
u32	szByte	Number of transfer bytes.	4

Table 3-9 NNSG2dUserExCellAttrBank

Type	Parameter Name	Description	Bytes
U16	numCells	Number of cells.	2
U16	numAttribute	Number of attributes per cell (currently fixed at 1).	2
NNSG2dUserExCellAtr	pCellAttrArray	Beginning of NNSG2dUserExCellAttr array.	4

Figure 3-2 Cell Data Bank General Structure
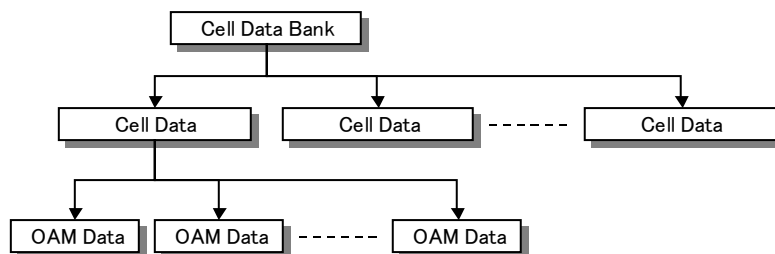
3.2 Description

3.2.1 NNSG2dCellDataBank.cellBankAttr

The `cellBankAttr` function stores the attribute information contained in cell data banks. It enumerates the currently stored information. Because the storage bits location and so on might change, use the accessor functions to access the information.

Cell data format (`NNSG2dCellData` or `NNSG2dCellDataWithBR`)

3.2.2 NNSG2dCellDataBank.mappingMode

`mappingMode` is the format information for the character data that is referenced by `Cell`. The following is the declaration of the enumeration type that defines the format information of the character data `NNSG2dCharacterDataMapingType`.

```
typedef enum NNSG2dCharacterDataMapingType
{
    NNS_G2D_CHARACTERMAPING_1D_32,
    NNS_G2D_CHARACTERMAPING_1D_64,
    NNS_G2D_CHARACTERMAPING_1D_128,
    NNS_G2D_CHARACTERMAPING_1D_256,
    NNS_G2D_CHARACTERMAPING_2D,
    NNS_G2D_CHARACTERMAPING_MAX

}NNSG2dCharacterDataMapingType;
```

3.2.3 NNSG2dCellDataBank.pVramTransferData

`NNSG2dCellDataBank` holds a pointer to the data required for implementing VRAM transfer animation in a member.

The cell definition data intended for drawing by the VRAM transfer has valid data set to `pVramTransferData`. (This is set to `NULL` if not used.)

By using this information, the VRAM transfer size and the transfer origin offset value for the character

data used in the cell can be obtained based on the cell number.

3.2.4 NNSG2dCellData

The NNSG2dCellData member cellAttr contains the OAM attribute information necessary for rendering as well as additional information. The information currently stored is listed below. Because the actual storage locations might change, use the accessor functions to access the information.

Currently stored information:

- Information about the flip feature usage status of the OBJs that make up the cell

- Information used to optimize cell rendering

- The radius of the bounding sphere that contains the cell

- Whether the cell has bounding rectangle information (if it can be cast to NNSG2dCellDataWithBR)

3.2.5 OAM Attribute Information

The OAM attribute data that is actually used for drawing is stored in the zone following the one in which NNSG2dCellData is stored. It is stored in a data format called NNSG2dCellOAMAttrData, which omits the affine parameter section.

The definition of NNSG2dCellOAMAttrData is as follows.

```
typedef struct NNSG2dCellOAMAttrData
{
u16 attr0;
u16 attr1;
u16 attr2;
}NNSG2dCellOAMAttrData;
```

In the file data, attribute data is stored in the form {attr0,attr1,attr2}, {attr0,attr1,attr2}, and so on. Because the end of the OAM attribute information string must be 4-byte aligned, the correct padding information must be inserted in the data.

3.2.6 User-Extended Attribute Information

User-extended attribute information is output by specifying the -oua option.

When this data is output, the offset address to the extended data is recorded in the member NNSG2dCellDataBank.pExtendedData. Extended data is managed by splitting it into individual blocks. Each data block includes a header that contains the block size and ID. The ID for the cell user-extended attribute block is given below.

```
NNS_G2D_USEREXBLK_CELLATTR        (u32)'UCAT'
```

4 NANR and NMAR (Animation Definition Information)

NANR (NMAR) is a file format that stores animation definition information. There are two types of animation: current cell animation and multi-cell animation. Both types of animation consist of the same data format. Only the contents of the `animType` member in `NNSG2dAnimSequence` are different. (A different name is defined in the runtime source to improve readability.) These formats are created by converting NCE and NMC NITRO-CHARACTER binary files. Animation definition information has a general block structure as shown below.

Table 4-1 NANR and NMAR Block Structure

Data Block Name	Identifier	Value	Condition	Notes
Animation Bank Block	NNS_G2D_BLKSIG_ANIMBANK	'ABNK'	Required	Specific
User-Extended Information Block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared
Name Label Information	NNS_G2D_BLKSIG_NAMELABEL	'LABL'	Optional	Shared

The animation definition information file's specific block is described in Figure 4-1.

4.1 Animation Bank Block

This block defines animation data. The general structure of this block is shown below.

Figure 4-1 NNSG2dCellBankBlock Structure
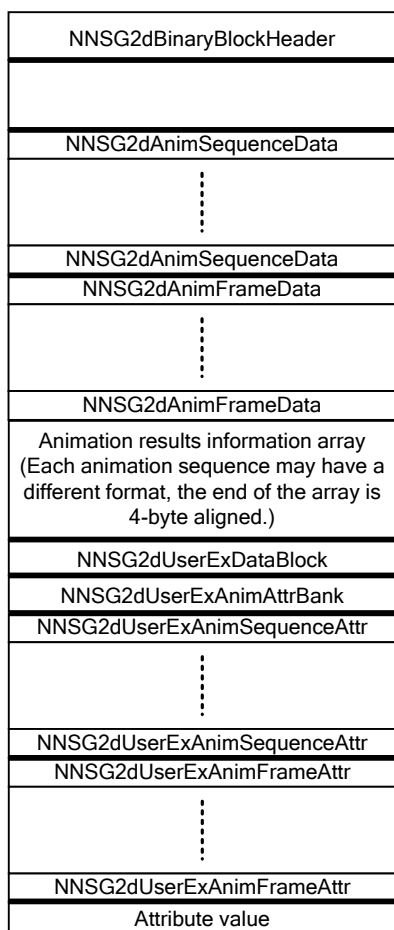
The following tables show the contents for each data structure.

Table 4-2 NNSG2dAnimBankDataBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header.	8
NNSG2dAnimBankData	cellDataBank	Animation frame standby frame count.	24

Table 4-3 NNSG2dAnimBankData

Type	Parameter Name	Description	Bytes
u16	numSequences	Number of animation sequence.	2
u16	numTotalFrames	Number of animation frame.	2
NNSG2dAnimSequenceData*	pSequenceArrayHead	Offset (pointer) to animation sequence information array.	4

Type	Parameter Name	Description	Bytes
NNSG2dAnimFrameData*	pFrameArrayHead	Offset (pointer) to animation frame information array.	4
void*	pAnimContents	Offset (pointer) to animation result array.	4
void*	pStringBank	Pointer to string bank (set during execution).	4
void*	pExtendedData	Pointer to library extended information (unused).	4

Table 4-4 NNSG2dAnimSequenceData

Type	Parameter Name	Description	Bytes
u16	numFrames	Number of animation frame.	2
u16	loopStartFrameIdx	Loop playback start frame index.	2
u32	animType	Animation result type.	4
NNSG2dAnimationPlayMode	playMode	Animation playback method.	4
NNSG2dAnimFrameData*	pAnmFrameArray	Offset (pointer) to animation frame array.	4

Table 4-5 NNSG2dAnimFrameData

Type	Parameter Name	Description	Bytes
void*	pContent	Offset (pointer) to animation results.	4
u16	frames	Animation frame standby frame count.	2
u16	pad16	Padding.	2

Table 4-6 NNSG2dUserExAnimAttrBank

Type	Parameter Name	Description	Bytes
u16	numSequences	Number of animation sequences.	2
U16	numAttribute	Number of attributes (fixed at 1).	2
NNSG2dUserExAnimSequenceAttr*	pAnmSeqAttrArray	Offset address (pointer) to the sequence attribute array.	2

Table 4-7 NNSG2dUserExAnimSequenceAttr

Type	Parameter Name	Description	Bytes
u16	numFrames	Number of animation frames.	2
u16	pad16	Padding.	2
u32*	pAttr	Pointer to array of attribute values.	4
NNSG2dUserExAnimFrameAttr*	pAnmFrmAttrArray	Offset address (pointer) to the frame attribute array.	4

Table 4-8 NNSG2dUserExAnimFrameAttr

Type	Parameter Name	Description	Bytes
u32*	pAttr	Array of attribute values.	4

Figure 4-2 Animation Data General Structure
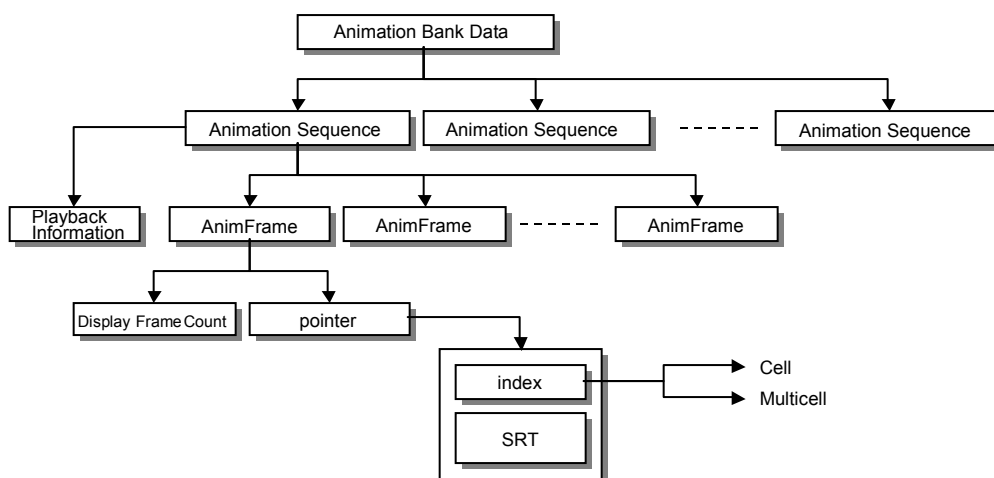
4.2 Description

4.2.1 NNSG2dAnimBankData.playMode

Stores information regarding the animation playback method. For playback method, one of four types is stored: one-time playback, repeat playback, round-trip playback, or round-trip repeat playback. The following is the declaration for the playback method enumerator NNSG2dAnimationPlayMode.

```
typedef enum NNSG2dAnimationPlayMode
{
    NNS_G2D_ANIMATIONPLAYMODE_INVALID = 0x0,   //Invalid
    NNS_G2D_ANIMATIONPLAYMODE_FORWARD,         //One-time playback
    NNS_G2D_ANIMATIONPLAYMODE_FORWARD_LOOP,    //Repeat playback
```

```
   NNS_G2D_ANIMATIONPLAYMODE_REVERSE,          //Round-trip playback

   NNS_G2D_ANIMATIONPLAYMODE_REVERSE_LOOP,    //Round-trip repeat playback

   NNS_G2D_ANIMATIONPLAYMODE_MAX

}

NNSG2dAnimationPlayMode;
```

4.2.2 NNSG2dAnimSequenceData.animType

NNSG2dAnimSequenceData.animType is a 32-bit value that stores information regarding the animation type. The lower 16 bits of NNSG2dAnimSequenceData.animType stores the animation result type. The animation results take multiple formats. For example, sequences that do not use SRT animation only have index stored in the animation results. By doing so, the data volume can be reduced. The following is the declaration for the animation result type enumeration NNSG2dAnimationElement.

```
typedef enum NNSG2dAnimationElement

{

   NNS_G2D_ANIMELEMENT_INDEX          = 0x0, // Index only

   NNS_G2D_ANIMELEMENT_INDEX_SRT          , // Index + SRT

   NNS_G2D_ANIMELEMENT_MAX

}

NNSG2dAnimationElement;
```

The upper 16 bits of NNSG2dAnimSequenceData.animType stores the animation type. Either cell animation or multi-cell animation is stored as the animation type. The following is the declaration for the animation type enumerated type NNSG2dAnimationType.

```
typedef enum NNSG2dAnimationType

{

   NNS_G2D_ANIMATIONTYPE_INVALID          = 0x0, // Illegal type

   NNS_G2D_ANIMATIONTYPE_CELL                  , // Cell

   NNS_G2D_ANIMATIONTYPE_MULTICELLLOCATION    ,// MultiCell

   NNS_G2D_ANIMATIONTYPE_MAX

}

NNSG2dAnimationType;
```

(Because specifications may change, we recommend the use of the dedicated accessor when accessing this data.)

4.2.3 Animation Results Array

Part of the animation results array stores the data that NNSG2dAnimFrameData.pContent points to. Several formats exist for representing animation results. You can reduce the amount of data by selecting the most suitable format for the information type used for each animation sequence. The animation results store data so that the end of the data has an alignment of 4 bytes for each animation sequence.

The specific format definition used for animation results is given below.

```
Typedef   u16      NNSG2dAnimData; // index only
Typedef   struct  NNSG2dAnimDataSRT      // index + SRT information
{
u16       index;  // index
u16       rotZ;   // rotation
fx32      sx;     // scale X
fx32      sy;     // scale Y


s16       px;     // position X
s16       py;     // position Y
}NNSG2dAnimDataSRT;


typedef   struct  NNSG2dAnimDataT // index + T information
{
u16       index;  // index
u16       pad_;   // rotation


s16       px;     // position X
s16       py;     // position Y
}NNSG2dAnimDataT;
```

4.2.4 User Extended Attribute Information

Specify the `-oua` option to get the output of the user-extended attribute information.

When this data has been output, the offset address to the extended data is stored in the member `NNSG2dAnimBank.pExtendedData`. Extended data is managed by splitting it into individual blocks. Each data block includes a header containing block size and ID. The ID for the user-extended attribute block cell is given below.

```
NNS_G2D_USEREXBLK_ANMATTR            (u32)'UAAT'
```

5 NCGR and NCBR (Character Information)

NCGR and NCBR are file formats that store character information. NCGR is arranged with the pixels grouped by character (8x8 pixels) (when the internal member `characterFmt` is `NNS_G2D_CHARACTER_FMT_CHAR`). NCBR is arranged with the pixels organized by picture element line (when the internal member `characterFmt` is `NNS_G2D_CHARACTER_FMT_BMP`). Both of these are generated by converting an NCG NITRO-CHARACTER binary file. The general block structure is shown below.

Table 5-1 NCGR (NCBR) Block Structure

Data Block Name	Identifier	Value	Condition	Notes
Character Definition Block	`NNS_G2D_BINBLK_SIG_CHARACTERDATA`	`'CHAR'`	Required	Fixed
Character Position Block	`NNS_G2D_BNIBLK_SIG_CHAR_POSITION`	`'CPOS'`	Optional	Fixed
User Extended Information Block	`NNS_G2D_BLKSIG_USEREXT`	`'UEXT'`	Optional	Shared

Below, the NCGR file's specific block is described.

5.1 Character Definition Block

The general structure for `NNSG2dCharacterDataBlock` is shown below.

Figure 5-1 NNSG2dCharacterDataBlock Structure
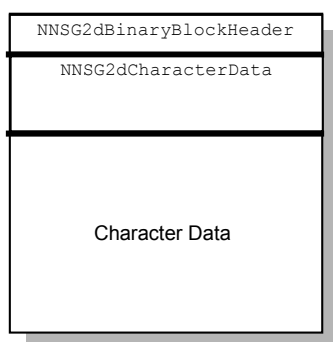
The following tables show the contents for each data structure.

Table 5-2 NNSG2dCharacterDataBlock

Type	Parameter Name	Description	Bytes
`NNSG2dBinaryBlockHeader`	`blockHeader`	Block header.	8
`NNSG2dCharacterData`	`characterData`	Character data.	24

Table 5-3 NNSG2dCharacterData

Type	Parameter Name	Description	Bytes
`u16`	`W`	Horizontal character count (2D map equivalent).	2
`u16`	`H`	Vertical character count (2D map equivalent).	2
`GXTexFmt`	`pixelFmt`	Pixel format.	4
`GXOBJVRamModeChar`	`mapingType`	Mapping type (1D or 2D).	4
`u32`	`characterFmt`	Character type. Character, bitmap, or VRAM transfer characters.	4
`u32`	`szByte`	Number of bytes for character data.	4
`void*`	`pRawData`	Offset pointer to character data.	4

5.2 Character Position Data Block

The basic configuration of `NNSG2dCharacterPosInfoBlock` is shown below.

Figure 5-2 NNSG2dCharacterPosInfoBlock Structure

NNSG2dBinaryBlockHeader
NNSG2dCharacterPosInfo
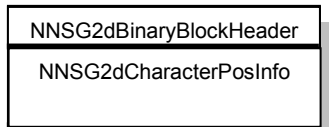
The contents of each data structure are listed in Table 5-4.

Table 5-4 NNSG2dCharacterPosInfo

Type	Parameter Name	Description	Bytes
`u16`	`srcPosX`	X position in extracted source character data (character units).	2
`u16`	`srcPosY`	Y position in extracted source character data (character units).	2
`u16`	`srcW`	Width of extracted source character data (character units).	4
`u16`	`srcW`	Height of extracted source character data (character units).	4

5.3 Description

5.3.1 NNSG2dCharacterData.W .H

NNSG2dCharacterData stores information about the format used to hold character data.

W and H store the size of the character units of the character data. However, this data is valid only when the character is stored in 2D mapping mode. For character data that is stored in 1D mapping mode, note that this data always records NNS_G2D_1D_MAPPING_CHAR_SIZE as a size that cannot be used for character data.

5.3.2 NNSG2dCharacterData.pixelFmt

pixelFmt is the picture element format for the character image. Only GX_TEXFMT_PLTT16 and GX_TEXFMT_PLTT256 are valid with G2D.

```
typedef enum
{
    GX_TEXFMT_NONE   = 0,
    GX_TEXFMT_A3I5   = 1,
    GX_TEXFMT_PLTT4  = 2,
    GX_TEXFMT_PLTT16 = 3,
    GX_TEXFMT_PLTT256 = 4,
    GX_TEXFMT_COMP4x4 = 5,
    GX_TEXFMT_A5I3   = 6,
    GX_TEXFMT_DIRECT  = 7
}
GXTexFmt;
```

5.3.3 NNSG2dCharacterData.characterFmt

characterFmt is an ordering type for the pixel data for the character data. The upper 8 bits store NNSG2dCharacterFmt, and the ninth bit is a flag that indicates whether it is a VRAM transfer character.

For the data drawn by using the 3D Graphics Engine (3DgraphicsEngin), the NNSG2dCharacterFmt value is NNS_G2D_CHARACTER_FMT_BMP. For the data drawn by using the 2D Graphics Engine (2DgraphicsEngin), the NNSG2dCharacterFmt value is NNS_G2D_Character_FMT_CHAR. Unused bits are reserved for future expansion.

Details of the enumerators stored in characterFmt are shown below.

```
typedef enum NNSG2dCharacterFmt
{
    NNS_G2D_CHARACTER_FMT_CHAR,
    NNS_G2D_CHARACTER_FMT_BMP,
    NNS_G2D_CHARACTER_FMT_MAX
}NNSG2dCharacterFmt;
```

5.3.4 NNSG2dCharacterData.mapingType

`mapingType` describes the type of character data stored. There are two major categories, 1D mode and 2D mode. See the *TWL Programming Manual* for details.

The enumerator definitions are shown below.

```
typedef enum
{
    GX_OBJVRAMMODE_CHAR_2D     ,
    GX_OBJVRAMMODE_CHAR_1D_32K ,
    GX_OBJVRAMMODE_CHAR_1D_64K ,
    GX_OBJVRAMMODE_CHAR_1D_128K,
    GX_OBJVRAMMODE_CHAR_1D_256K
}
GXOBJVRamModeChar;
```

5.3.5 NNSG2dCharacterPosInfoBlock

The character position information block is output only when the `-cr/` option has been specified during binary conversion of character data in 2D mapping mode.

6 NCLR (Color Palette Information)

NCLR is a file format that stores palette information. It is created by converting an NCL NITRO-CHARACTER binary file. NCLR has a general block structure as shown below.

Table 6-1 NCLR Block Structure

Data Block Name	Identifier	Value	Condition	Notes
Palette Definition Block	NNS_G2D_BINBLK_SIG_PALETTEDATA	'PLTT'	Required	Fixed
Palette Compression Information Block	NNS_G2D_BINBLK_SIG_PALETTECOMPINFO	'PCMP'	During compression	Fixed
User-Extended Information Block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared

The NCLR file's specific block is described below.

6.1 Palette Definition Block

The general structure for NNSG2dPaletteDataBlock is shown below:

Figure 6-1 NNSG2dPaletteDataBlock Structure
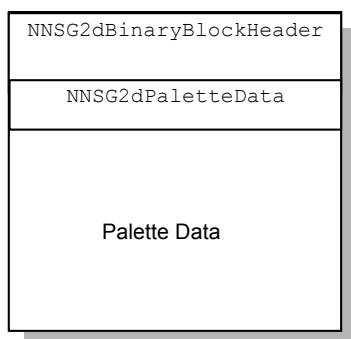
The following tables show the contents for each data structure.

Table 6-2 NNSG2dPaletteDataBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	8
NNSG2dPaletteData	paletteData	Palette data	16

Table 6-3 NNSG2dPaletteData

Type	Parameter Name	Description	Bytes
GXTexFmt	fmt	Palette format	4
BOOL	bExtendedPlt	Whether the Extended Palette is used	4
u32	szByte	Number of bytes for character data	4
void*	pRawData	Offset pointer to character data	4

6.2 Palette Compression Information Block

The general structure of the NNSG2dPaletteCompressDataBlock is shown below.

Figure 6-2 NNSG2dPaletteCompressDataBlock Structure
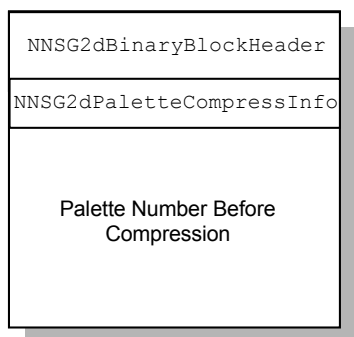
The content of each data structure is shown below.

Table 6-4 NNSG2dPaletteCompressDataBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	8
NNSG2dPaletteCompressInfo	plttCmpInfo	Palette compression data	8

Table 6-5 NNSG2dPaletteCompressInfo

Type	Parameter Name	Description	Bytes
u16	numPalette	Number of palettes	2
u16	pad16	Padding	2
void*	pPlttIdxTbl	Palette number array before compression	4

6.3 Description

6.3.1 NNSG2dPaletteCompressInfo

A compressed palette memorizes data only for the palette numbers that are in use. This data is used to restore the data to its original state. The `NNSG2dPaletteCompressInfo`, `pPlttIdxTbl`, points to a u16 palette number array having an array length equal to the number of palettes after compression. This array is used as a palette number conversion table to extract the original palette numbers.

7 NMCR (MultiCell Information)

NMCR is a file format that stores multi-cell definition information. It is created by converting an NMC NITRO-CHARACTER binary file (not supported in the current version of the converter). The general block structure is shown below.

Table 7-1 NMCR Block Structure

Data Block Name	Identifier	Value	Condition	Notes
MultiCell Bank Block	NNS_G2D_BLKSIG_MULTICELLBANK	'MCBK'	Required	Specific
User-Extended Information Block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared
Name Label Information	NNS_G2D_BLKSIG_NAMELABEL	'LABL'	Optional	Shared

The NMCR file's specific block is described below.

7.1 Multi-Cell Bank Block

The general structure for NNSG2dMultiCellDataBankBlock is shown below.

Figure 7-1 NNSG2dMultiCellDataBankBlock Structure
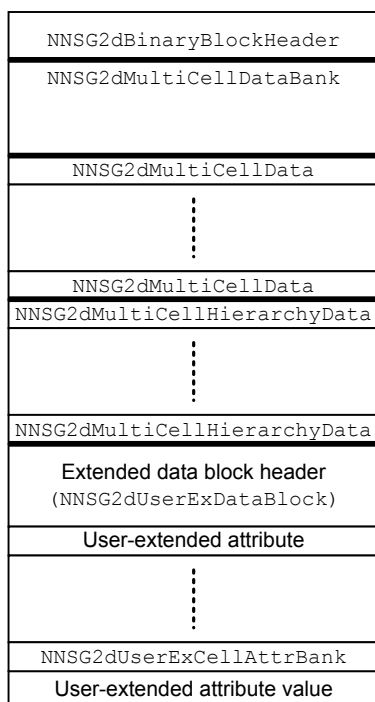
The following tables show the contents for each data structure.

Table 7-2 NNSG2dMultiCellDataBankBlock Structure

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	2
NNSG2dMultiCellDataBank	multiCellDataBank	Multi-cell definition data	20

Table 7-3 NNSG2dMultiCellDataBank

Type	Parameter Name	Description	Bytes
u16	numMultiCellData	Number of multi-cell data sets	2
u16	pad16	Padding	2
NNSG2dMultiCellData*	pMultiCellDataArray	Offset (pointer) to the multi-cell definition information array	4
NNSG2dMultiCellHierarchyData*	pHierarchyDataArray	Offset (pointer) to the cell animation element setting information array	4
void*	pStringBank	Pointer to string bank (set during execution)	4
void*	pExtendedData	Pointer to library extended information (unused)	4

Table 7-4 NNSG2dMultiCellData

Type	Parameter Name	Description	Bytes
u16	numNodes	Number of cell animation nodes that constitute the multi-cell	2
u16	numCellAnim	The minimum number of cell animation entities needed	2
NNSG2dMultiCellHierarchyData*	pHierDataArray	Offset (pointer) to the cell animation element setting information array	4

Table 7-5 NNSG2dMultiCellHierarchyData

Type	Parameter Name	Description	Bytes
u16	animSequenceIdx	Playback animation sequence number	2
s16	posX	Position X	2
s16	posY	Position Y	2
u16	nodeAttr	Node attribute (NNSG2dMCAminationPlayMode and so on.)	2

Figure 7-2 MultiCell Definition Data General Structure
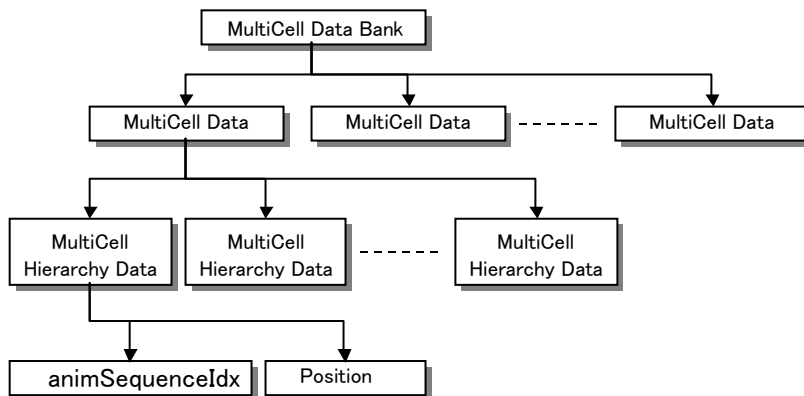
7.2 Description

7.2.1 NNSG2dMultiCellHierarchyData. nodeAttr

`nodeAttr` stores the animation play mode for cell animation bound to a node and visibility flag information.

The lower 4 bits store `NNSG2dMCAnimationPlayMode`. The fifth bit stores a visibility flag. Bits 16-8 store the cell animation numbers of the minimum required cell animations.

For example, when there are 10 nodes and they all play the same cell animation, the minimum necessary cell animation is one, and all `nodeAttr` store 0 as the cell animation number.

This information is not used in the current runtime library.

Unused bits are reserved for future expansion.

The definition for `NNSG2dMCAnimationPlayMode` is given below.

```
typedef enum NNSG2dMCAnimationPlayMode
{
    NNS_G2D_MCANIM_PLAYMODE_RESET     = 0,
    NNS_G2D_MCANIM_PLAYMODE_CONTINUE  = 1,
    NNS_G2D_MCANIM_PLAYMODE_PAUSE     = 2,
    NNS_G2D_MCANIM_PLAYMODE_MAX

}NNSG2dMCAnimationPlayMode;
```

7.2.2 User Extended Attribute Information

User extended attribute information stored in the multi-cell bank uses the exact same data format as the cell bank. For this reason, the data format is not described.

8 NENR (Entity Information)

NENR is a file format that stores entity definition information. Although leaning more toward the user program, the data that requires library support is planned to be stored in this structure. (Currently, it has only playback animation sequence number information.) NENR is generated by converting entity definition text using the Win32 application `BuildNENR.exe`. NENR has the general block structure shown below.

Table 8-1 NENR Block Structure

Data Block Name	Identifier	Value	Condition	Notes
Entity Bank Block	NNS_G2D_BLKSIG_ENTITYBANK	'ENBK'	Required	Specific
User-Extended Information Block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared
Name Label Information	NNS_G2D_BLKSIG_NAMELABEL	'LABL'	Optional	Shared

The NENR file's specific block is described below.

8.1 Entity Bank Block

The general structure for `NNSG2dEntityDataBankBlock` is shown below.

Figure 8-1 NNSG2dEntityDataBankBlock Structure
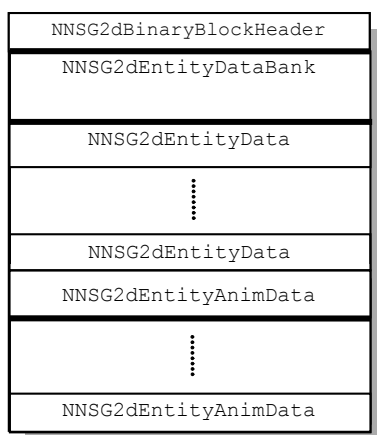
The following tables show the contents for each data structure.

Table 8-2 NNSG2dEntityDataBankBlock Structure

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	2
NNSG2dEntityDataBank	entityDataBank	Entity definition data bank	20

Table 8-3 NNSG2dEntityDataBank

Type	Parameter Name	Description	Bytes
u16	numEntityDatas	Number of Entity data sets.	2
u16	pad16	Padding.	2
NNSG2dEntityData*	pEntityDataArray	Offset pointer to Entity data array.	4
u16*	pAnimSequenceIdxArray	Offset pointer to animation sequence number array.	4
void*	pStringBank	Pointer to string bank (set during execution).	4
void*	pExtendedData	Pointer to library extended information (unused).	4

Table 8-4 NNSG2dEntityData

Type	Parameter Name	Description	Bytes
NNSG2dEntityAnimData	animData	Animation information	8
NNSG2dEntityType	type	Entity type (Cell or MultiCell)	4

Table 8-5 NNSG2dEntityAnimData

Type	Parameter Name	Description	Bytes
u16	numAnimSequence	Animation sequence number count.	2
u16	pad16	Padding.	2
u16*	pAnimSequenceIdxArray	Offset pointer to animation sequence number array.	4

Figure 8-2 Entity Data General Structure
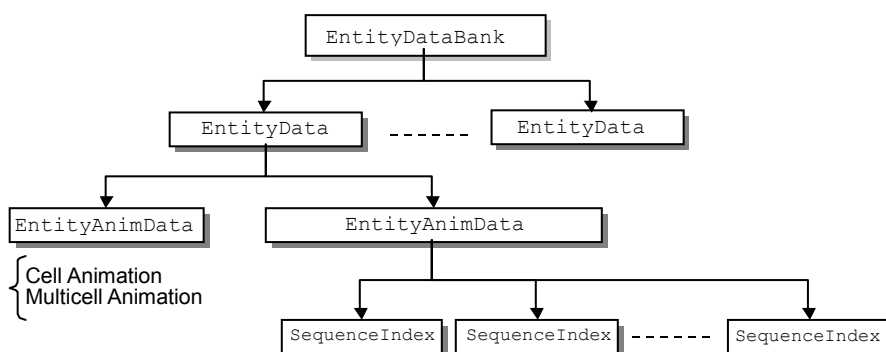
9 NSCR (Screen Information)

NSCR is a file format that stores screen information. It is created by converting NITRO-CHARACTER binary files that have the .nsc extension.

The general block structure of these files is shown here.

Table 9-1 NSCR Block Structure

Data Block Name	Identifier	Value	Condition	Notes
Screen Definition Block	NNS_G2D_BINBLK_SIG_SCRDATA	'SCRN'	Required	Fixed
User-Extended Information Block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared

The NSCR file's specific block is described in the following section.

9.1 Screen Definition Block

The general structure of the NNSG2dScreenDataBlock is shown below.

Figure 9-1 NNSG2dScreenDataBlock Structure
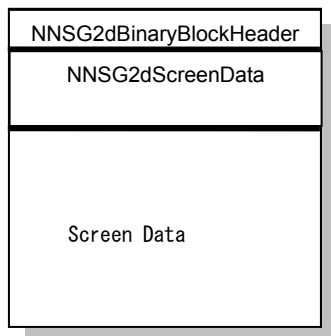
The following tables show the contents for each data structure.

Table 9-2 NNSG2dScreenDataBlock

Type	Parameter	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	8
NNSG2dScreenData	screenData	Screen data	12

Table 9-3 NNSG2dScreenData

Type	Parameter Name	Description	Bytes
u16	screenWidth	Screen width in pixels	2
u16	screenHeight	Screen height in pixels	2
u16	colorMode	Supported palette format	2
u16	screenFormat	BG type	2
u32	szByte	Screen data byte count	4
u8[]	rawData	Screen data	szByte

9.2 Description

9.2.1 NNSG2dScreenData. ColorMode

colorMode is the color palette type used for the display of screen data.

Details of the enumerator stored in colorMode are shown below.

```
typedef enum NNSG2dColorMode
{
    NNS_G2D_SCREENCOLORMODE_16x16,
    NNS_G2D_SCREENCOLORMODE_256x1,
    NNS_G2D_SCREENCOLORMODE_256x16
}
NNSG2dColorModel;
```

9.2.2 NNSG2dScreenData. screenFormat

screenFormat is the BG type of the stored screen data.

Details of the enumerator stored in screenFormat are shown below.

```
typedef enum NNSG2dScreenFormat
{
    NNS_G2D_SCREENFORMAT_TEXT,
    NNS_G2D_SCREENFORMAT_AFFINE,
    NNS_G2D_SCREENFORMAT_AFFINEEXT
}
NNSG2dScreenType;
```

10 NFTR (Font Information)

This chapter describes the NFTR file format, which is a format that stores font data. Files with this format are created by converting BMP files or Windows fonts using `fontcvtr.exe`. The general block structure of these files is as follows.

Table 10-1 NFTR Block Configuration

Data Block Name	Identifier	Value	Condition	Notes
Font information block	NNS_G2D_BINBLK_SIG_FINFDATA	'FINF'	Required	Fixed
Glyph image block	NNS_G2D_BINBLK_SIG_CGLPDATA	'CGLP'	Required	Fixed
Character width block	NNS_G2D_BINBLK_SIG_CWDHDATA	'CWDH'	Optional	Fixed
Character code map block	NNS_G2D_BINBLK_SIG_CMAPDATA	'CMAP'	Optional	Fixed
User-extended information block	NNS_G2D_BLKSIG_USEREXT	'UEXT'	Optional	Shared

A single font information block and glyph information block are required. However, there can be any number of other blocks or none at all.

The NFTR file's fixed block is described below.

10.1 Font Information Block

The font information block (`NNSG2dFontInformationBlock`) holds information about the font. By resolving the offset pointer during runtime, other blocks can be accessed by pointers using the font information block as a reference.

The general structure of the `NNSG2dFontInformationBlock` is as follows.

Figure 10-1 NNSG2dFontInformationBlock Structure

```
NNSG2dBinaryBlockHeader

NNSG2dFontInformation
```
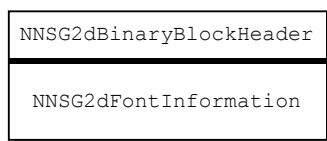
The contents of each data structure are shown in the following tables.

Table 10-2 NNSG2dFontInformationBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block header	8
NNSG2dFontInformation	blockBody	Font information	20

Table 10-3 NNSG2dFontInformation

Type	Parameter Name	Description	Bytes
u8	fontType	Font type	1
u8	linefeed	Font line feed width	1
u16	alterCharIndex	Alternate character glyph index	2
NNSG2dCharWidths	defaultWidth	Default character width information	3
u8	encoding	Supported string encoding	1
NNSG2dFontGlyph*	pGlyph	Offset pointer to glyph image	4
NNSG2dFontWidth*	pWidth	Offset pointer to character width data	4
NNSG2dFontCodeMap*	pMap	Offset pointer to character code map data	4
u8	height	Font height	1
u8	width	Font width (not used in the current library)	1
u8	ascent	Ascent	1
u8	padding_	Region for padding	1

Table 10-4 NNSG2dCharWidths

Type	Parameter Name	Description	Bytes
s8	left	Character's left space width	1
u8	glyphWidth	Character's glyph width	1
s8	charWidth	Character's width	1

10.2 Glyph Image Block

The glyph image block (NNSG2dFontGlyphBlock) stores the glyph image and glyph image data for each character, which is the core information for the font.

The general structure of the NNSG2dFontGlyphBlock is as follows.

Figure 10-2 NNSG2dFontGlyphBlock Structure

The contents of each data structure are shown in the following tables.

Table 10-5 NNSG2dFontGlyphBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block data	8
NNSG2dFontGlyph	blockBody	Glyph data	8

Table 10-6 NNSG2dFontGlyph

Type	Parameter Name	Description	Bytes
u8	cellWidth	Cell width	1
u8	cellHeight	Cell height	1
u16	cellSize	Cell data size	2
s8	baselinePos	Baseline position from the top of the cell	1
u8	maxCharWidth	Maximum character width	1
u8	bpp	No. of bits per pixel in cell	1
u8	reserved	(Reserved)	1
u8[]	glyphTable	Glyph data (cell array)	Variable

10.3 Character Width Block

The character width block (NNSG2dFontWidthBlock) stores the character widths of the glyph images that are stored in the glyph image block. When several character width blocks are stored in a single font resource, they are organized in a linked list. If a glyph does not have a character width corresponding to the character width block, the default width defined in the font information block is applied.

The general structure of the NNSG2dFontWidthBlock is as follows.

Figure 10-3 NNSG2dFontWidthBlock Structure
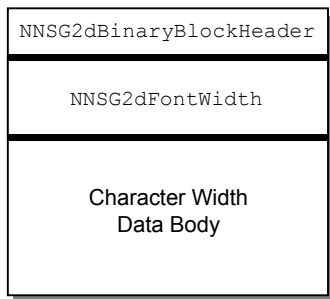
The contents of each data structure are shown in the following tables.

Table 10-7 NNSG2dFontWidthBlock

Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block data	8
NNSG2dFontWidth	blockBody	Character width data	8

Table 10-8 NNSG2dFontWidth

Type	Parameter Name	Description	Bytes
u16	indexBegin	Glyph index value for the first entry	2
u16	indexEnd	Glyph index value for the last entry	2
NNSG2dFontWidth	pNext	Offset pointer to the next NSG2dFontWidth	4
NNSG2dCharWidths[]	widthTable	Array of character width data	Variable

10.4 Character Code Map Block

The character code map block (NNSG2dFontCodeMapBlock) stores correspondences between character codes and glyph images. A single font resource generally has several character code map blocks, and these are organized as a linked list. To obtain the corresponding glyph index from the character code, follow the linked list in order from the character information block, and use the first convertible block for conversion. This allows the block at the end of this linked list to include the range covered by blocks in front of it on the list, and this prevents the breakup of blocks.

The general structure of NNSG2dFontCodeMapBlock is as follows.

Figure 10-4 NNSG2dFontCodeMapBlock Structure

```
┌─────────────────────────────┐
│  NNSG2dBinaryBlockHeader     │
├─────────────────────────────┤
│                             │
│     NNSG2dFontCodeMap        │
│                             │
├─────────────────────────────┤
│                             │
│    Character Code Map        │
│       Data Body              │
│                             │
└─────────────────────────────┘
```
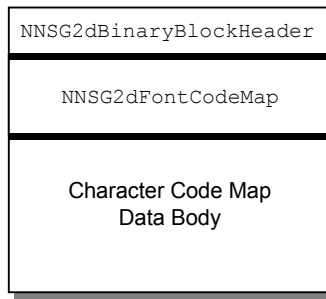
The contents of each data structure are shown in the following tables.

Table 10-9 NNSG2dFontCodeMapBlock

| Type	Parameter Name	Description	Bytes
NNSG2dBinaryBlockHeader	blockHeader	Block data	8
NNSG2dFontCodeMap	blockBody	Font code map data	12

Table 10-10 NNSG2dFontCodeMap

Type	Parameter Name	Description	Bytes
u16	ccodeBegin	Start of char codes convertible by this block character codes	2
u16	ccodeEnd	End of char codes convertible by this block	2
u16	mappingMethod	Character code mapping method	2
u16	reserved	(Reserved)	2
NNSG2dFontCodeMap*	pNext	Offset to next NNSG2dFontCodeMap	4
u16	mapInfo	Table of character codes and glyph index	Variable

Table 10-11 NNSG2dCMapInfoScan

Type	Parameter Name	Description	Bytes
u16	num	Number of elements in entries	2
NNSG2dCMapScanEntry[]	entries	Table of char codes and glyph index values	Variable

Table 10-12 NNSG2dCMapScanEntry

Type	Parameter Name	Description	Bytes
u16	ccode	Character code	2
u16	index	Glyph index value	2

Figure 10-5 General Structure of Font Data
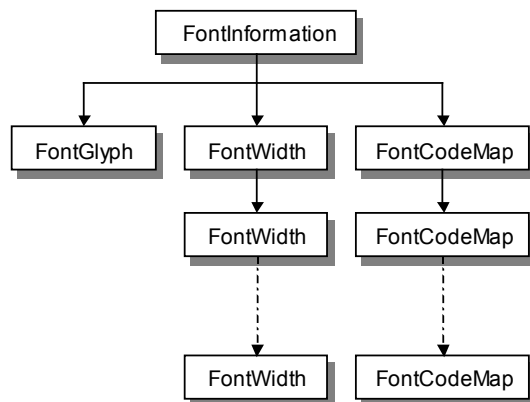
10.5 Description

10.5.1 NNSG2dFontInformation.fontType

`NNSG2dFontInformation.fontType` stores the glyph information type of the font resource. Currently, only `NNS_G2D_FONTTYPE_GLYPH` is defined.

10.5.2 NNSG2dFontInformation.pGlyph, pWidth, and pMap

These are offset pointers to the initial block of the various block types. Each block contains the offset pointer to the next block, forming a one-way linked list. If there is no corresponding block or the block is at the end of the list, `NULL` is stored.

By resolving each block's offset pointer at runtime, all blocks in the font resource can be accessed by pointer from the font information block.

10.5.3 NNSG2dFontGlyph.flags

This is a collection of flags that express the font's characteristics. Flags that indicate fonts for vertical writing and vertical-style are stored.

10.5.4 NNSG2dFontGlyph.glyphTable

`NNSG2dFontGlyph.glyphTable` is an array of the bitmap data in which glyph images are stored. Each bitmap data has the size indicated by `cellSize`, and all bitmap data is stored continuously without any alignment. Bitmap data is stored in ascending order by character code.

10.5.5 NNSG2dFontWidth.indexBegin, indexEnd, and widthTable

`indexBegin` stores the glyph index value corresponding to the first element in the `widthTable` and `indexEnd` stores the glyph index value corresponding to the last element. In other words, the character width information for the glyph with an index value of `indexBegin` is stored as the first element of the `widthTable`.

Because width information for all glyph index values is stored in order in the `widthTable` from `indexBegin` to `indexEnd`, you can obtain the character width information for the desired glyph index value by using [Glyph index value - `indexBegin`] as the subscript of the array. The number of character width information elements stored in the `widthTable` is [`indexEnd` - `indexBegin` + 1].

10.5.6 NNSG2dFontCodeMap.ccodeBegin and ccodeEnd

`ccodeBegin` and `ccodeEnd` indicate the beginning and end of the character codes that are handled by the block in question. They do not indicate that glyphs for all of the character codes from `ccodeBegin` and `ccodeEnd` are stored in the font resource.

10.5.7 NNSG2dFontCodeMap.mappingMethod and mapInfo

`NNSG2dFontCodeMap.mappingMethod` specifies the method used to convert character codes to glyph index values. The information stored in `mapInfo` changes depending on the conversion method. The following three conversion methods are defined.

- `NNS_G2D_MAPMETHOD_DIRECT`

  The glyph index value is determined from the character code by using the following calculation. `MapInfo` stores a single glyph index offset of type u16.

  Glyphs are stored for all the character codes from `ccodeBegin` to `ccodeEnd`.

  ```
  Glyph index value = character code – ccodeBegin + glyph index offset
  ```

- `NNS_G2D_MAPMETHOD_TABLE`

  The glyph index value is determined by looking up the character code as a key in the table. The number of glyph index values stored in `MapInfo` as type `u16` are [`ccodeEnd` - `ccodeBegin` + 1]. If the obtained glyph index value is 0xFFFF, it indicates that no corresponding glyph is stored.

  ```
  Glyph index value = MapInfo[character code – ccodeBegin]
  ```

- `NNS_G2D_MAPMETHOD_SCAN`

  Searches from the array of (character code/glyph index value) pairs using the character code as a key. In this case, `MapInfo` stores data of type `NNSG2dCMapInfoScan`. Because elements in `NNSG2dCMapInfoScan`.entries are stored in ascending order of character code, a binary search can be used. If the character code is not found, it indicates that the corresponding glyph is not stored.

Windows is a registered trademark or trademark of Microsoft Corporation (USA) in the U.S. and other countries.

All other company and product names in this document are the trademarks or registered trademarks of their respective companies.