NitroROM File System Specifications

NITRO-SDK

10/19/04
SPD Environment Design Group, Nintendo Co., Ltd.

0 Introduction

This document introduces a simple file system in the NitroSDK that functions as a method to shorten, as much as possible, the time to update the data within ROM that frequently occurs during development.

This document also describes how the program files and data files that are generated during the application creation with Nitro are made into a ROM file.

You do not need to be concerned about this format from applications because ROM can be accessed through a file system-related API.

There is also a possibility that changes may occur in the final product version concerning details such as registration addresses of data that are mentioned in this document.

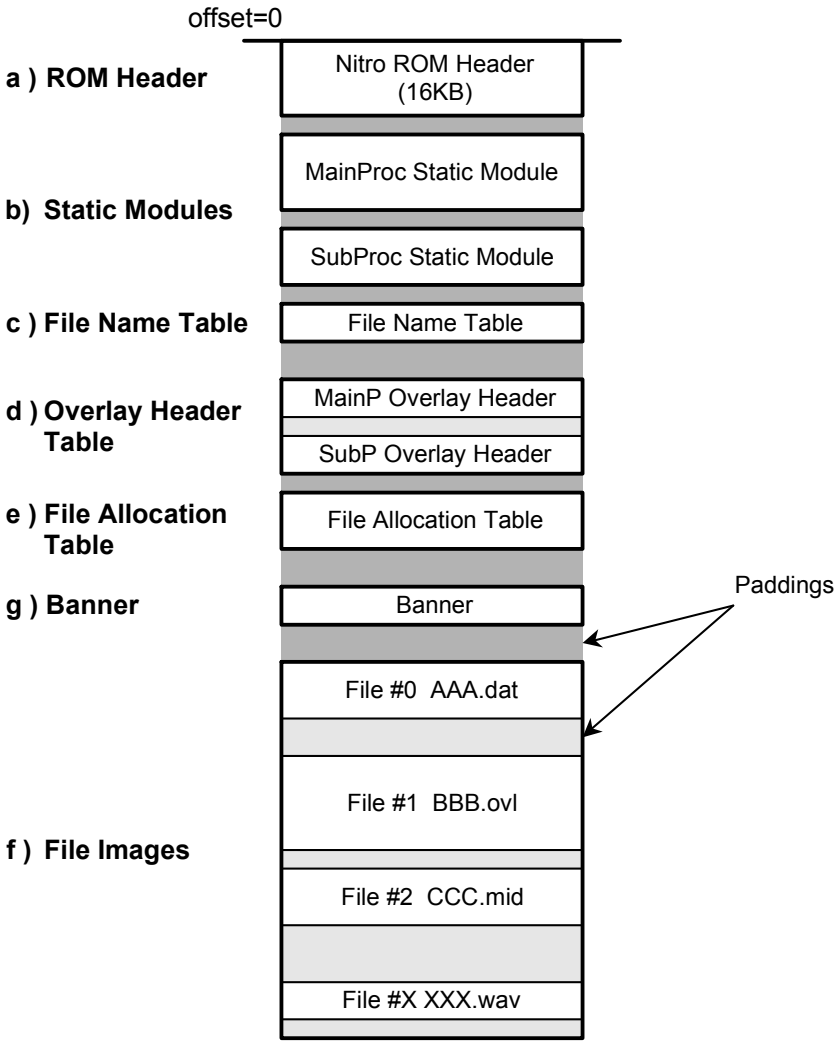
The changes made since 08/04/2004 are shown in red.

1 NitroROM Format

A Nitro ROM file consists of the following programs:

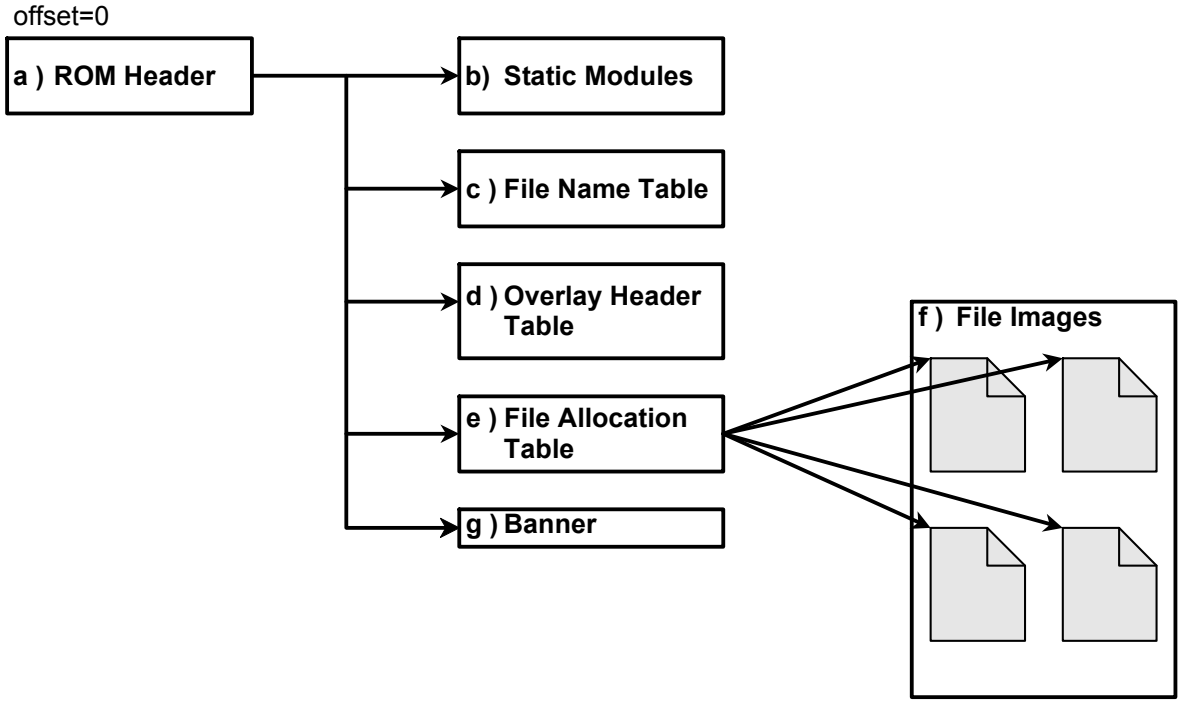
(a)	ROM header	Management data for the entire ROM
(b)	Static module (MainP/SubP)	Module that is read and executed at startup
(c)	File name table	Corresponding information between file names and file numbers
(d)	Overlay header table	Corresponding information between overlay ID and file numbers
(e)	File allocation table	Location information of each file within the ROM (correspondence between file numbers and locations)
(f)	File image	File entity
(g)	Banner	Banner file. Stores icons and game names

NTR-06-0037-002-A1
Released: February 2, 2006



If the alignment between each block and the start addresses of sections within blocks is required, padding will be inserted. The developer can specify unit length settings and whether or not to align for each block.

In the illustration above, addresses are allocated in order of a) - e), g, f). This illustration, however, is only for the purpose of easily understanding the organization. With the exception of a), block positions will not be the same order. The ROM header is fixed at a top offset within the ROM. Other blocks are linked from the ROM header by pointers (offset values from the top of the ROM) directly or indirectly. Because of this, you can freely change the arrangement of blocks, except for the ROM header, by changing the order of links. The following illustration is a conceptual link diagram. This is for reducing the costs that are related to emulations of ROM images.



The C source is used to show the organization of each NitroROM block.

(a) ROM Header

```
typedef struct
{
   //
   // 0x000  Reserved region for system use
   //
   u8       reserved_A[32];           // Reserved for system A (not explained in this text)
   //
   // 0x020 b)  Parameter for static module
   //
   //       ARM9
   void*    main_rom_offset;          // Source ROM offset
   void*    main_entry_address;       // Execution start address (not implemented)
   void*    main_ram_address;         // Destination RAM address
   u32      main_size;                // Size
   //       ARM7
   void*    sub_rom_offset;           // Source ROM offset
   void*    sub_entry_address;        // Execution start address (not implemented)
   void*    sub_ram_address;          // Destination RAM address
   u32      sub_size;                 // Size
```

```
//
// 0x040 c)  Parameter for file name table
//
ROM_FNTDir* fnt_offset;   // Top ROM offset
u32         fnt_size;     // Table size
//
// 0x048 e)  Parameter for file allocation table
//
ROM_FAT* fat_offset;               // Top ROM offset
u32      fat_size;                 // Table size
//
// 0x0050 d)  Parameter for overlay header table
//
//       ARM9
ROM_OVT* main_ovt_offset;          // Top ROM offset
u32      main_ovt_size;            // Table size

//       ARM7
ROM_OVT* sub_ovt_offset;           // Top ROM offset
u32      sub_ovt_size;             // Table size
//
// 0x0060 - 0x0067 Reserved region for system use
//
u8       reserved_B1[8];           // Reserved for system B1 (not explained in this text)
//
// 0x0068 g) Banner file offset
//
u32      banner_offset;    // Top ROM offset
//
// 0x006c - 0x006f Reserved for system
//
u8       reserved_B2[4];   // Reserved for system B2 (not explained in this text)
//
// 0x0070 Static module parameter 2 (for debugger)
//
void*    main_autoload_done;       // ARM9 AUTOLOAD complete CALLBACK
void*    sub_autoload_done;        // ARM7 AUTOLOAD complete CALLBACK
//
// 0x0078 - 0x03fff Reserved for system
```

```
  //
u8 reserved_C[4*1024-0x78];  // Reserved for system C
u8 reserved_D[12*1024];      // Reserved for system D

} ROM_Header;                          // 16KB
```

(b) Static module (MainP/SubP)

This module is output as a binary file at the same time as elf files during link processing. This binary file is inserted in ROM as is.

There are two processors: the main processor ARM9 and the sub processor ARM7. The top ROM address of each processor must be aligned at 512 bytes.

(c) File name table

This table acquires file IDs from file names. It supports directories. The table consists of a directory table and an entry name table.

The directory table has the following structure array.

A number referred to as a directory ID is assigned to each table. The directory ID is incremented in the order of the stored data. To distinguish it from the directory ID, file IDs start from 0xF000. The maximum value is 0xFFFF. From these specifications, the maximum number of directories is 4,096 and the maximum number of files is 61,440.

The number of elements in the array coincides with the number of directories; the subscript of the array coincides with the value of 0xF000 subtracted from the directory ID.

Data that has a directory ID of 0xF000 represents a root directory. For a root directory, the number of directory entries is stored in the members of the parent directory ID.

```
typedef struct
{
  u32      entry_start;     // Search location of entry name
  u16      entry_file_id;   // File ID of top entry
  u16      parent_id;       // ID of parent directory
} ROM_FNTDir;
```

`entry_start` (search location of entry name) is an offset value from the top location of a file name table.

A file name table is a collection of the following two types of variable length data. Use the data structures correctly depending on whether the entries are files or directories. Since there is a need for processing these data structures in units of bytes, you must be careful of restrictions on byte access when analyzing this data in main memory.

```
typedef struct
{
  u8       entry_type       :1;      // 0 when file entry
  u8       entry_name_length:7;      // Length of file name (0 - 127)
  char     entry_name[length];       // File name (omit terminal \0)
} ROM_FNTStrFile;
typedef struct
{
  u8       entry_type       :1;      // 1 when directory entry
  u8       entry_name_length:7;      // Length of directory name (0  - 127)
  char     entry_name[length];       // Directory name (omit terminal \0)
  u8       dir_id_L;                 // Directory ID Low 8-bit
  u8       dir_id_H;                 // Directory ID High 8-bit
} ROM_FNTStrDir;
```

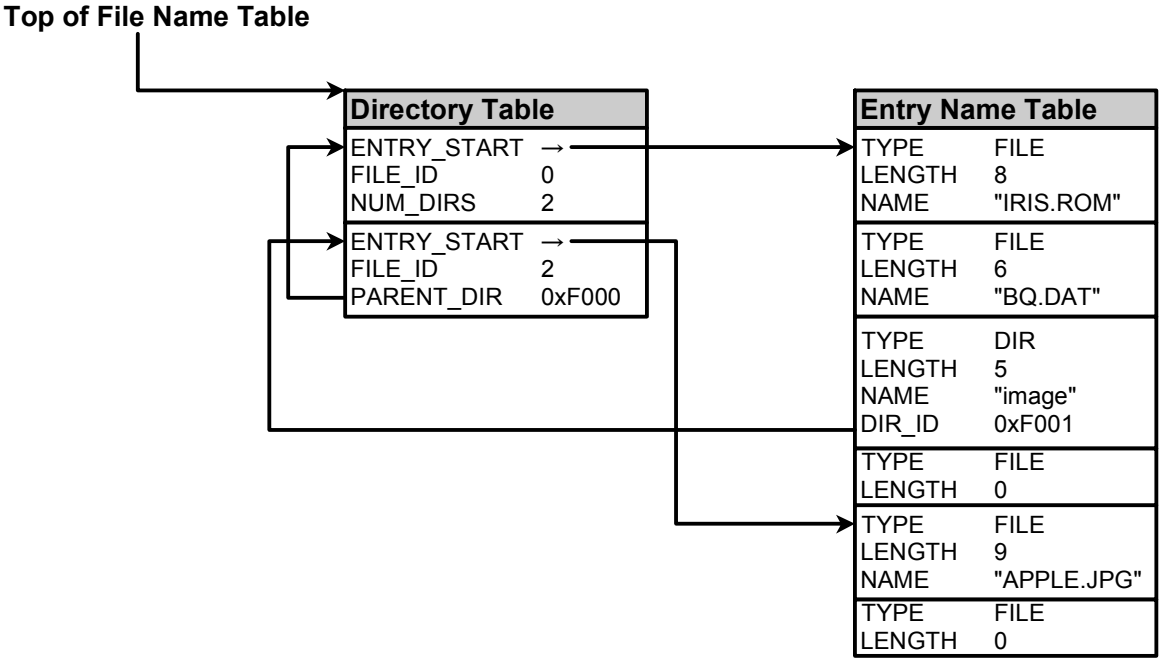
Entries that are included in identical directories are arranged in successive regions. With the exception of “files” that are subdirectories contained in the directories, “ files” are assigned successive file IDs. File entries (“\0”) that have an entry name length of 0 are placed after the final entry within a directory.

Entry names have the following characteristics:

- A maximum of 127 characters (in terms of 1-byte character).
- Distinguish between uppercase and lowercase characters with the specification of file names in order to speed up the search process.
- The registration of multiple entries with identical names within the same directory is prohibited. Taking into consideration that work will be performed in Windows, uppercase and lowercase characters are not distinguished when judging whether or not there are identical entry names being registered.
- You can use characters for the entry names except for code that cannot be used with Windows within the range of ASCII 0x20 – 0x7e (`\ / : ; * ? " < > |`). The internationalization of file names is not supported because of support cost considerations.

The following illustration shows an example when the three following file names are stored in this format.

```
/Nitro.ROM
/BQ.DAT
/image/APPLE.JPG
```



(d) Overlay header table

This is a file that contains load information of overlay files. It is created as a binary file at the same time with nef files and overlay modules during link processing.

When a linker outputs, the “overlay file ID” is set to a temporary value. The “overlay file ID” is then rewritten with an actual value by the `makerom` command.

The following is an array of the structure data. The size of the array coincides with the number of overlay files, and the subscript of the array coincides with the overlay ID.

```
typedef struct
{
   u32      id;              // Overlay ID
   void*    ram_address;     // Load top location
   u32      ram_size;        // Load size
   u32      bss_size;        // bss region size
   void*    sinit_init;      // static initializer initial address
   void*    sinit_init_end;  // static initializer end address
   u32      file_id;         // Overlay file ID
   u32      reserved;        // Reserved (0 is set)
   u32      compressed:24;    // Overlay size after compressed
   u32      flag      :8;     // Overlay information flag

} ROM_OVT;
```

Note) Regarding the bit field of compressed and flag, address +0 is set to the area for flag, and address +1 to +3 is set to the area for compressed.

To support the compression of overlay files, the overlay information flag was newly added. The OR values with the following values are set according to the overlay state. This value is evaluated by the library when the overlay is loaded.

Compressed 0x01
Authentication code included 0x02

09/17/04 flag region was moved. Compressed was added.
09/04/04 flag region was added.

03/29/04 The ROM_OVT overlay file size value has been scrapped. It is now reserved.

(e) File allocation table

A file allocation table has an array of the structure data shown below. A number, called a file ID, is assigned to each table. The file ID is incremented in the order of storage from 0x0000. The maximum value is 0xEFFF. The size of the array coincides with the number of files and the subscript of the array that coincides with the file ID.

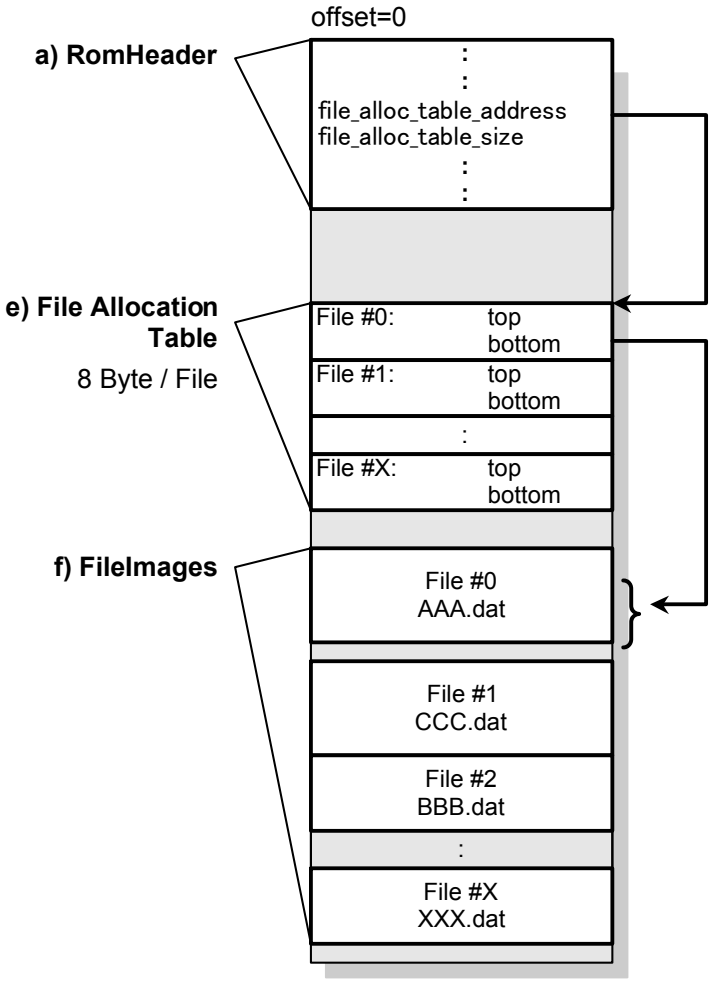
```
typedef struct
{
   void*    head;    // Top ROM address of file
   void*    tail;    // Bottom ROM address of file
} ROM_FAT;           // 0x08
```

In the specification of after 02/17/04, reservation of the 0 value for the upper 4 bits of `ROM_FilePtr` was canceled.

When the file ID is specified at intervals, { 0, 0 } is used for the file allocation table region that corresponds to an unused file ID.

(f) File image

Each file corresponds to each entry of the file allocation table. The file is placed in a region that is specified by an address between the file top and the file bottom.



g) Banner File

The banner file stores the images and messages displayed on the selection screen immediately after the game starts. The starting ROM address must be 512-byte aligned.

```
typedef struct
{
   //       Header
   u8       version;                  //  The current version is 0x01
   u8       reserved_A;
   u16      crc16_v1;                 // CRC for checking
   u8       reserved_B[28];
} BannerHeader;                       // 32B
typedef struct
```

```
{
  //       Icon data         H32xW32x16colors
  u8       image[32*32/2];           // 32 * 32 * 4bit    = 512B
  u8       pltt[16*2];               // 16color * 16bit   = 32B
  //       Game name data    Encoding:UTF16-LE(without BOM)
  u16      gameName[6][128];         // 6langs * 128chars = 1536B
} BannerFileV1;                      // 2080B
typedef struct
{
    BannerHeader  h;
    BannerFileV1  v1;
} BannerFile;                        // 2112B
```

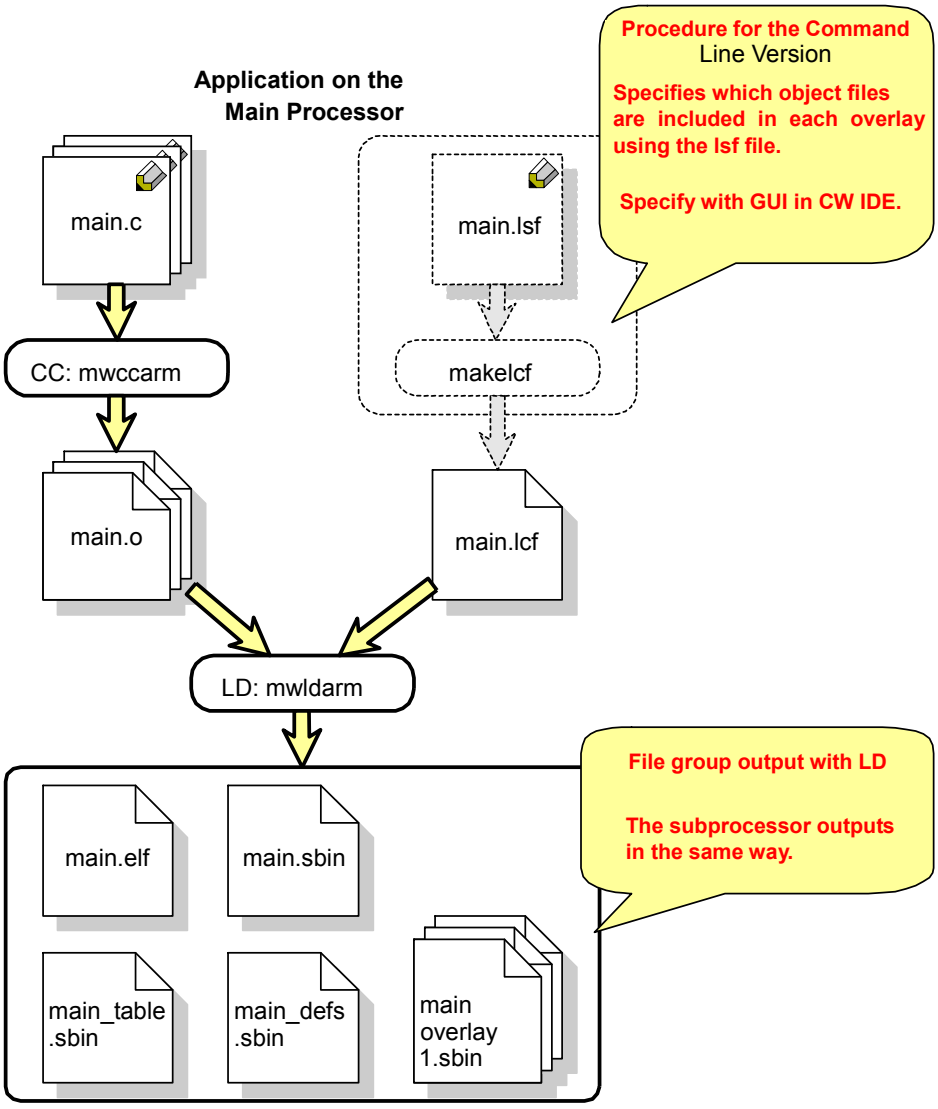
09/29/04 The banner file format explanation was added.

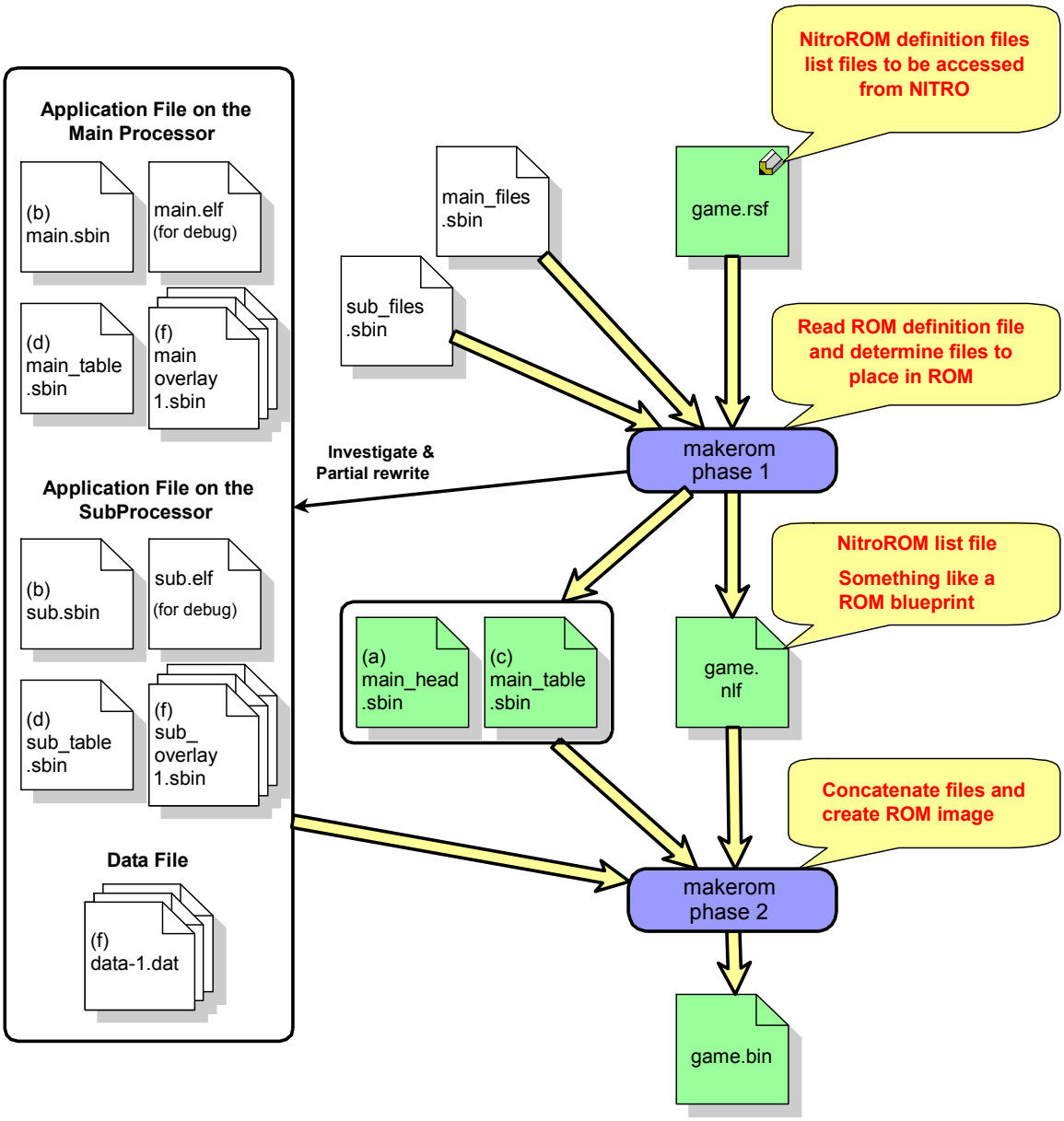
2 NitroROM Creation Path

ROM files are generated in paths as shown below.

The application called makerom have important functions during ROM file creation. `makerom` performs the following processes:

- Phase 1
  - Determines the files to be placed within the ROM, determines the offset location of each file within the f) File image block (previously described), and then outputs that information as a file.
  - A complete form of “c) File name table” and a model of “a) ROM header” are created simultaneously with the processes described in the first bullet.
  - Adds more information to “d) Overlay header table” to make a complete form.
- Phase 2
  - Generates “e) File Allocation Table” and “f) File Image Block” based on the information file that is output in phase 1.
  - Adds information to “a) ROM header” to make a complete form.
  - Links all of a) to f) and makes this a ROM file.


Application on the
Main Processor
main.c
CC: mwccarm
main.o
main.lsf
makelcf
main.lcf
LD: mwldarm
main.elf
main.sbin
main_table
.sbin
main_defs
.sbin
main
overlay
1.sbin
Procedure for the Command
Line Version
Specifies which object files are included in each overlay using the lsf file.
Specify with GUI in CW IDE.
File group output with LD
The subprocessor outputs in the same way.



`makerom` uses the following setting files.

(a) NitroROM spec file extension `.rsf`

(b) NitroROM list file extension `.nlf`

These files are described here.

(a) **NitroROM spec file .rsf**

This is an example of a definition file. For details on the format, see the makerom tool reference manual.

Describe the binary file portion (`.sbin`) output by the linker to Arm9/Arm7 section and then adds the files you want to add within the ROM file to the `RomSpec` section. Also, describe other information in the `Property` section.

In the following example, a file with a `.jpg` extension within the `data/graphics` directory of a development PC is added under the `/data` directory of the ROM file. Then, in the next region a file with a `.wav` extension under `data/ARM7/sound` is added under the `/sound` directory.

```
#
#  NitroROM Spec File
#
Arm9
{
   Static            main.sbin
   OverlayDefs       main_defs.sbin
   OverlayTable      main_table.sbin
   Nef               main.nef
}
Arm7
{
   Static            sub.sbin
   OverlayDefs       sub_defs.sbin
   OverlayTable      sub_table.sbin
   Nef               sub.nef
}
Property
{
   RomHeader         main_head.sbin
   FileName          main_files.sbin
   BannerFile        bannerfile.sbin
}
RomSpec
{
   Offset   0x00000000
   Segment  ALL
```

```
  Align    512
  Padding  0xff

  HostRoot data/graphics
  Root              /data
  File              *.jpg
  HostRoot data/ARM7/sound
  Root              /sound
  File              *.wav
}
```

(b) **NitroROM list file .nlf**

This is a CSV format file that includes an information set that is used to build ROM images. This file determines the ambiguous parts of rsf files. An example is shown below.

```
#NLF --- NitroROM List File
V,1.1
T,"C:/NitroSDK/build/tests/file/file-1"
H,"rom_header.bin","rom_files.bin",15
9,"main.nef","main.bin","main_ovt.bin","main_ovn.bin","."
7,"sub.nef","sub.bin","sub_ovt.bin","sub_ovn.sbin","."
# File Image Block
F,00000000,000201fc,00,001c,ffff, "ROMROOT/Nitro.ROM","/Nitro.ROM",3ffbf36e,512,0
P,000201fc,00000200,00
.....
```

Line feed code is `\r\n`.

Each line consists of a command that consists of one ASCII character as the first parameter, and parameters after that command.

Commands are divided into two types; A header command that indicates overhead information and a body command that specifies the actual contents of the ROM. A body command should not be positioned in front of a header command.

Header Command

```
Version:   [V]
  V,[version number of file]
```

This is the version number of the format of a ROM file. Changed to 1.1 because the information for the relative path from the overlay binary top directory has been added to `arm9files/arm7files`.

```
Topdir:    [T]
  T,[Top directory name]
```

This is a reference directory in the indirect path designation expression of a file. The current directory is set when an NLF file is created. If the top directory is designated with a relative path, the position of the top directory is interpreted using the directory where the NLF file is located as reference.

```
Headers:   [H]
  H,[ROM header file],[File name table file],[Number of files with file IDs]
```

Specify the file names of ROM header files and file name tables.

These files are created by `makerom`. The file names are enclosed by double quotes.

A value of 8 times the `[Number of files with file IDs]` becomes the size of the file allocation table.

```
arm9files: [9]
  9,[nef file],[Static module bin file],
            [Overlay table file],
            [Overlay name file],[relative path from overlay binary top directory]
```

Note: We have put this on separate lines to make it easier to see. However, it is actually one line.

Specifies the file name of the application file that is used for the Main Processor.

Each file name is enclosed by double quotes.

You can obtain the access path to binaries in the overlay name table by connecting `[top directory]` and `[relative path from overlay binary top directory]`.

If the overlay table file is not needed, set "*" as a file name. At this time * is also set to `[relative path from overlay binary top directory]`.

```
arm7files: [7]
  7,[nef file],[Static module bin file],
            [Overlay table file],
            [Overlay name file],[relative path from overlay binary top directory]
```

Note: We have put this on separate lines to make it easier to see. However, it is actually one line.

Specifies the file name of the application file that is used for the Sub Processor.

Each file name is enclosed by double quotes.

If the overlay table file is not needed, set "`*`" as a file name. At this time * is also set to `[relative path from overlay binary top directory]`.

Body Command

```
File:               [F]
  F,[Start offset],[End offset],[Padding code],
                    [File ID],[ROM header offset],
                    [File name in development machine],[ROM file name],[Time stamp of file],
                    [Alignment value],[Movement prohibit flag 0: Move permitted  1: Prohibit]
```

Note: Line feeds are entered for clarity; the command should be on one line.

Adds files to file image blocks as ROM files.

ROM address space where target files are placed is the range of addresses shown below:

Start offset ≤ address < End offset

When a file is smaller than the specified address space and the range that is specified by the start offset and end offset cannot be filled, the remaining region will be filled with padding code. In contrast, when the file is larger than the address space, an error will occur.

Stores information that is related to the offset location of files within ROM in a file allocation table according to the specified value of [File ID]. The storage location is a location that is [File ID] * 8 from the top of the file allocation table. Refer to the description of the file allocation table in the preceding section for details on the storage structure. In addition, the specified value need not be stored when it is `ffff`.

In the same way, the file whose [ROM header offset] is not set to ffff stores offset information of files at specified locations of the ROM header. The storage location is a location that is [ROM header offset] * 4 from the top of the ROM header. Refer to the description of the ROM header for details.

The [File name in development machine] shows the locations of files in a development PC.

The file allocation table is generated when executing makerom. Because of this, the file allocation table does not exist as a file on the PC. A special file name, `*FILEALLOC`, is defined to specify the insertion location of the file allocation table.

Except for ordinary files, file names are not set for data files that are related to file system building such as an overlay table. An asterisk (*) is specified in the [ROM file name] field of files without names. This measure is taken to simplify the analysis of command character strings using sscanf.

The [Time stamp of file] is a `time_t` type value (total number of seconds from 1970 UTC) of a C library as well as the value of a member `st_mtime` of a stat structure that is acquired by the stat function. This field is set to 0 for a file allocation table (`*FILEALLOC`).

Changing the values of the start offset / end offset is not allowed for files whose [Movement prohibit flag] is set to a value of “1.”

When this flag is “0”, the offset can be changed. When changing the offset value, be sure to change each offset using the numeric value units that are specified by the alignment value.

The format of each value is as follows: (`sscanf` notation)

Command	`%1c`
Start offset	`%08x`
End offset	`%08x`
Padding code	`%02x`
File ID	`%04x`
ROM header offset	`%04x`
File name in development machine	`\"%1024[^\"\n]\"`
ROM file name	`\"%1024[^\"\n]\"`
Time stamp of file	`%08x`
Alignment value	`%d`
Movement prohibit flag	`0` or `1`

```
Padding:   [P]
  P,[Start offset],[End offset],[Padding code]
```

Adds padding to the ROM file image block.

The target ROM address space is the following range of addresses:

Start offset ≤ address < End offset

The range that is specified by the start offset and the end offset is filled with padding code.

```
[Other]
#comment:  [#]
  # [Comment]
```

None. For comments. Comma-delimited is not necessary for this line.

```
#NLF
```

The starting four characters of the file are `#NLF` in consideration of use as a magic number for a file.

3 Overlay process

In order to obtain the overlay parameter at the time of `makerom` execution, you must create an overlay name file in the format below when linking.

This file saves startup parameters (16-byte) that are related to the resident module and the executable binary filenames of each overlay generated during linking.

The overlay executable binary file names are packed in the order of overlay IDs as character strings terminated with "\0". For example, when file names are a.sbin, b.sbin, and c.sbin, the file name data is saved in the following format:

```
a.sbin\0b.sbin\0c.sbin\0
```

To obtain the file name of overlay with overlay ID of "N", search for the Nth "\0" from the start of the file name data, and get the character string that starts from the next character as the file name.

```
//
// OverlayDefs format
//
typedef struct ROM_ONTHeader
{
    void*  static_ram_address;       // static module ram_address
    void*  static_entry_address;     //               entry address
    u32    static_size;              //               size
    void*  static_autoload_done,     // static autoload done address (debug purpose)
} ROM_ONTHeader;
typedef struct ROM_ONT
{
    ROM_ONTHeader   header;
    char            file_list[];     // Variable length STRING data
                                     // File names ending in a NULL character are
                                     // retained as many as numbers corresponding to the
                                     // overlay binary
} ROM_ONT;
```