NITRO ROM ファイルシステム仕様

NITRO-SDK

2004/10/19

任天堂株式会社 環境制作グループ

0. はじめに

NITRO-SDK では開発時に頻繁に発生する ROM 内のデータの更新作業を出来るだけ短時間で行なうための手法として簡易なファイルシステムを導入いたします。
本ドキュメントは Nitro でのアプリケーション作成において生成されるプログラムファイルおよびデータファイルをどのようにして ROM 化されるかを解説するものです。

ROM へのアクセスにはファイルシステム関連のAPI を通じて行うことができるため、アプリケーションからこのフォーマットを気にする必要はありません。

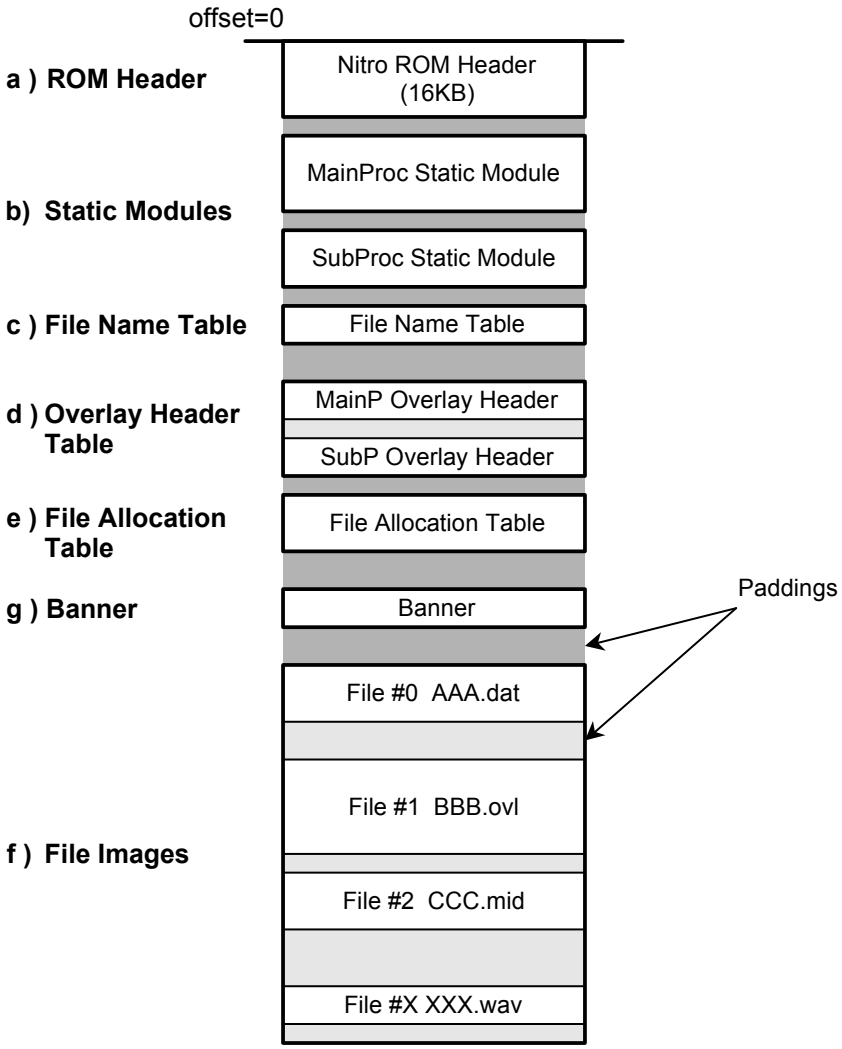
また本仕様書において記載されているデータの登録アドレスなどの詳細部については最終製品版において変更される可能性があります。

04/08/04 版からの変更点は赤文字になっています。

1. NitroROM フォーマット

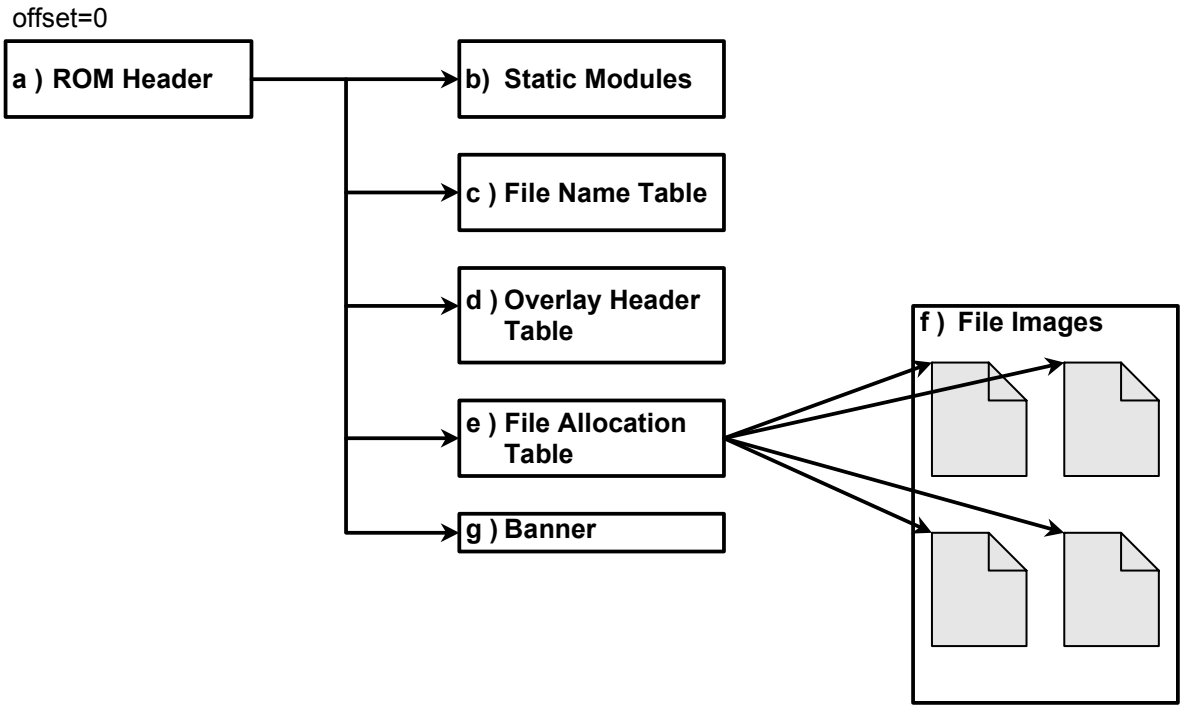
Nitro の ROM ファイルは下記のようなブロックから構成されています。

a) ROM ヘッダ – ROM 全体の管理データ
b) 常駐モジュール（MainP/SubP） – 起動時に読み込まれ最初に実行されるモジュールです
c) ファイル名テーブル – ファイル名とファイル番号との対応情報
d) オーバーレイヘッダテーブル – オーバーレイ ID ファイル番号との対応情報
e) ファイルアロケーションテーブル – 各ファイルのROM内位置情報（ファイル番号と位置との対応）
f) ファイルイメージ – ファイルの実体
g) バナー – バナーファイル。アイコンやゲーム名称が格納される



各ブロックやブロック内のセクションの開始アドレスでアライメントを取る必要がある場合はパディングが挿入されます。アライメントの有無および単位長の設定はブロック毎に開発者が指定できます。

上図では a) ～ e), g), f) の順にアドレスが割り振られていますが、これはあくまで構造をわかりやすくするための便宜上のもので、a) 以外のブロックの位置は順不同です。ROM ヘッダは ROM 内の先頭オフセットに固定されていますが、その他のブロックは直接または間接に ROM ヘッダからポインタ(ROM 先頭からのオフセット値)によりリンクされています。このため ROM ヘッダ以外のブロックはリンクを付け替えることによって配置を自由に変更することができます。下図がリンクの概念図です。これにより ROM イメージのエミュレーションについてのコストの削減を狙っています。



NitroROM ブロックのそれぞれの構造を Cソースで示します。

a) ROM ヘッダ

```
typedef struct
{
        //
        // 0x000 システム予約領域
        //
        u8          reserved_A[32];             // システム予約 A (本稿では説明
略)

        //
        // 0x020 b) 常駐モジュール用パラメータ
        //
        //          ARM9
        void*       main_rom_offset;            // 転送元 ROM オフセット
        void*       main_entry_address;         // 実行開始アドレス(未実装)
        void*       main_ram_address;           // 転送先 RAM アドレス
        u32         main_size;                  // 転送サイズ

        //          ARM7
        void*       sub_rom_offset;             // 転送元 ROM オフセット
        void*       sub_entry_address;          // 実行開始アドレス(未実装)
```

```
    void*       sub_ram_address;            // 転送先 RAM アドレス
    u32         sub_size;                   // 転送サイズ

    //
    // 0x040 c) ファイルネームテーブル用パラメータ
    //
    ROM_FNTDir* fnt_offset;                 // 先頭 ROM オフセット
    u32         fnt_size;                   // テーブルサイズ

    //
    // 0x048 e) ファイルアロケーションテーブル用パラメータ
    //
    ROM_FAT*    fat_offset;                 // 先頭 ROM オフセット
    u32         fat_size;                   // テーブルサイズ

    //
    // 0x0050 d) オーバーレイヘッダテーブル用パラメータ
    //
    //          ARM9
    ROM_OVT*    main_ovt_offset;            // 先頭 ROM オフセット
    u32         main_ovt_size;              // テーブルサイズ

    //          ARM7
    ROM_OVT*    sub_ovt_offset;             // 先頭 ROM オフセット
    u32         sub_ovt_size;               // テーブルサイズ

    //
    // 0x0060 - 0x0067 システム予約域
    //
    u8          reserved_B1[8];             // システム予約 B1 (本稿では説明
略)

    //
    // 0x0068 g) バナーファイルオフセット
    //
    u32         banner_offset;              //  先頭 ROM オフセット

    //
    // 0x006c - 0x006f システム予約域
    //
    u8          reserved_B2[4];             // システム予約 B2 (本稿では説明
略)

    //
    // 0x0070 常駐用モジュールパラメータ 2 (デバッガ用)
    //
    void*       main_autoload_done;         // ARM9 AUTOLOAD 完了
CALLBACK
```

```
      void*       sub_autoload_done;          // ARM7 AUTOLOAD 完了
CALLBACK

      //
      // 0x0078 - 0x03fff システム予約域
      //
      u8          reserved_C[4*1024-0x78];    // システム予約 C
      u8          reserved_D[12*1024];        // システム予約 D

} ROM_Header;     // 16KB
```

b) 常駐モジュール (MainP/SubP)

リンク処理時に elf ファイルと同時にバイナリファイルとして出力されます。
このバイナリファイルがそのまま ROM に挿入されます。

メインプロセッサ ARM9 側のものとサブプロセッサ ARM7 側のものとの2つがあり、おのおのの先頭 ROM アドレスは 512 バイトアラインメントされていなければなりません。

c) ファイル名テーブル

ファイル名からファイルID を取得するためのテーブルです。ディレクトリをサポートします。
ディレクトリテーブルとエントリ名テーブルから構成されます。

ディレクトリテーブルは下記の構造体の配列です。
各テーブルにはディレクトリID と呼ばれる番号が割り振られています。ディレクトリID は格納順にカウントアップされます。ファイルID と区別するためディレクトリID は 0xF000 から開始します。最大値は 0xFFFF です。この仕様より、ディレクトリ、ファイルの最大数はそれぞれ 4096 ディレクトリ、61440 ファイルとなります。

配列の要素数はディレクトリの数と一致し、配列の添え字がディレクトリID から 0xF000 を減じた値と一致します。
ディレクトリID が 0xF000 であるデータはルートディレクトリを示します。ルートディレクトリの場合、親ディレクトリ ID のメンバにディレクトリエントリ数が格納されます。

```
typedef struct
{
      u32         entry_start;                // エントリ名の検索位置
      u16         entry_file_id;              // 先頭エントリのファイル ID
      u16         parent_id;                  // 親ディレクトリの ID
} ROM_FNTDir;
```

entry_start (エントリ名の検索位置)は、ファイル名テーブルの先頭位置からのオフセット値です。

エントリ名テーブルは下記の2種類の可変長のデータの集合です。そのエントリがファイルであるかディレク

トリであるかによってデータ構造を使い分けます。これらのデータ構造ではバイト単位での処理を行なう必要性が高いためメインメモリ上でこのデータを解析するときはバイトアクセス制限に気をつける必要があります。

```
typedef struct
{
      u8          entry_type      :1;          // ファイルエントリの場合は 0
      u8          entry_name_length:7;         // ファイル名の長さ (0-127)
      char        entry_name[length];          // ファイル名 (終端 ¥0 は省く)
} ROM_FNTStrFile;

typedef struct
{
      u8          entry_type      :1;          // ディレクトリエントリの場合は 1
      u8          entry_name_length:7;         // ディレクトリ名の長さ (0-127)
      char        entry_name[length];          // ディレクトリ名 (終端 ¥0 は省く)
      u8          dir_id_L;                    // ディレクトリ ID Low  8bit
      u8          dir_id_H;                    // ディレクトリ ID High 8bit
} ROM_FNTStrDir;
```

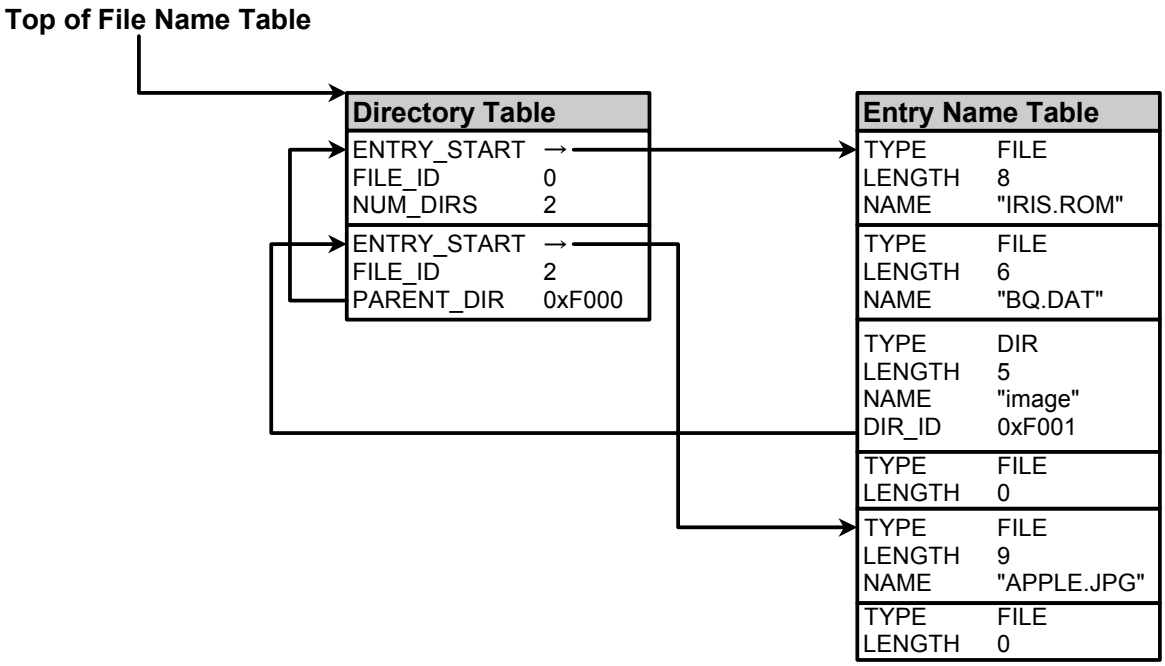
同一ディレクトリ内に含まれるエントリは連続した領域に配置されており、その中に含まれるサブディレクトリを除いた「ファイル」には連続したファイルIDが割り振られます。ディレクトリ内の最終エントリの次にはエントリ名の長さが0であるファイルエントリ('¥0') が置かれます。

エントリ名は

- 最大 127 文字(1byte文字換算)
- 検索の処理の高速化のため、ファイル名の指定において大文字と小文字を区別します。
- 同一ディレクトリ内に同じ名前のエントリを複数登録することは禁止します。ただし Windows 上で作業を行なうことを鑑みて、このエントリ内への登録における同一かどうかの判定では大文字と小文字を区別しません。
- エントリ名は ASCII コード 0x20-0x7e から Windows で使用できないコード(¥ / : ; * ? " < > \|) を除いた文字のみ使用できます。またファイル名の国際化は対応のコストを考慮し非対応とします。

下図は以下の3つのファイル名が本フォーマットで格納された場合の模式図です。

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



d) オーバーレイヘッダテーブル

オーバーレイファイルのロード情報を含んだファイルです。リンク処理時に nef ファイルやオーバーレイモジュールと同時にバイナリファイルとして作成されます。

リンカが出力した時点では［オーバーレイファイルID］は仮の値が設定されています。［オーバーレイファイルID］は makerom コマンドにより実際の値に書き換えられます。

下記の構造体データの配列となっています。配列のサイズはオーバーレイファイル数と一致し、配列の添え字がオーバーレイIDと一致します。

```
typedef struct
{
      u32         id;                     // オーバーレイ ID
      void*       ram_address;            // ロード先頭位置
      u32         ram_size;               // ロードサイズ
      u32         bss_size;               // bss 領域サイズ
      void*       sinit_init;             // static initializer 先頭アドレス
      void*       sinit_init_end;         // static initializer 最終アドレス
      u32         file_id;                // オーバーレイファイルID
      u32         compressed:24;          // オーバーレイ圧縮後のサイズ
      u32         flag      :8;           // オーバーレイ情報フラグ
} ROM_OVT;
```

注) compressed と flag のビットフィールドは、アドレス +0 に flag が、アドレス +1～+3 に compressed

のエリアが設定されるものとする。

オーバーレイファイルの圧縮などに対応するために、オーバーレイ情報フラグを新設しました。この領域にはオーバーレイの状態に応じて以下の値のOR値が設定されます。この値はオーバーレイをロードするときにライブラリで評価されます。

圧縮済み	0x01
認証コード収録済み	0x02

04/09/17 flag 領域が移動、compressed が追加されました。
04/09/04 flag 領域が追加されました。
04/03/29 ROM_OVT のオーバーレイファイルサイズ値は廃止され、reserved になりました。

e) ファイルアロケーションテーブル

ファイルアロケーションテーブルは下記の構造体データの配列となっています。各テーブルにはファイルIDと呼ばれる番号が割り振られています。ファイルID は格納順に 0x0000 からカウントアップされます。最大値は 0xEFFF です。配列のサイズはファイル数と一致し、配列の添え字がファイルIDと一致します。

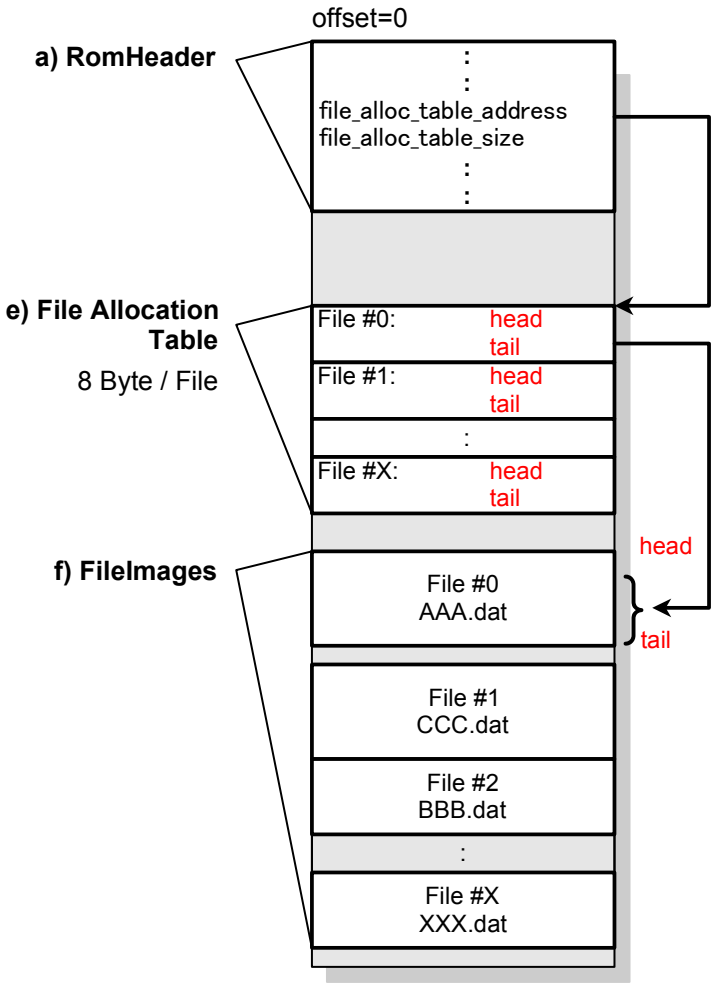
```
typedef struct
{
      void*           head;          // ファイルの先頭 ROM アドレス
      void*           tail;          // ファイルの最終 ROM アドレス
} ROM_FAT;            // 0x08
```

04/02/17 以降の仕様では ROM_FilePtr の上位 4bit の 0 値の予約は解除されました。

ファイルID が飛び飛びに指定されている場合、使用されていないファイルID に対応するファイルアロケーションテーブル領域には { 0, 0 } が入ります。

f) ファイルイメージ

各ファイルはファイルアロケーションテーブルの各エントリと対応し、ファイルはファイル先頭と最終とのアドレスで指定された領域に置かれています。



g) バナーファイル

バナーファイルはゲームスタート直後の選択画面で表示されるイメージおよびメッセージを格納するためのファイルです。先頭 ROM アドレスは 512 バイトアラインメントされていなければなりません。

```
typedef struct
{
      //              ヘッダ
      u8              version;              // 現バージョンは 0x01
      u8              reserved_A;
      u16             crc16_v1;             // チェック用 CRC
      u8              reserved_B[28];
} BannerHeader;       // 32B

typedef struct
{
      //              アイコンデータ     H32xW32x16colors
      u8              image[32*32/2];       // 32 * 32 * 4bit    = 512B
      u8              pltt[16*2];           // 16color * 16bit   = 32B
```

```
      //              ゲーム名データ    Encoding:UTF16-LE(without BOM)
      u16             gameName[6][128];    // 6langs * 128chars = 1536B
} BannerFileV1;       // 2080B

typedef struct
{
   BannerHeader  h;
   BannerFileV1  v1;
} BannerFile;         // 2112B
```

04/09/29 バナーファイルフォーマットの説明の項目が追加されました。

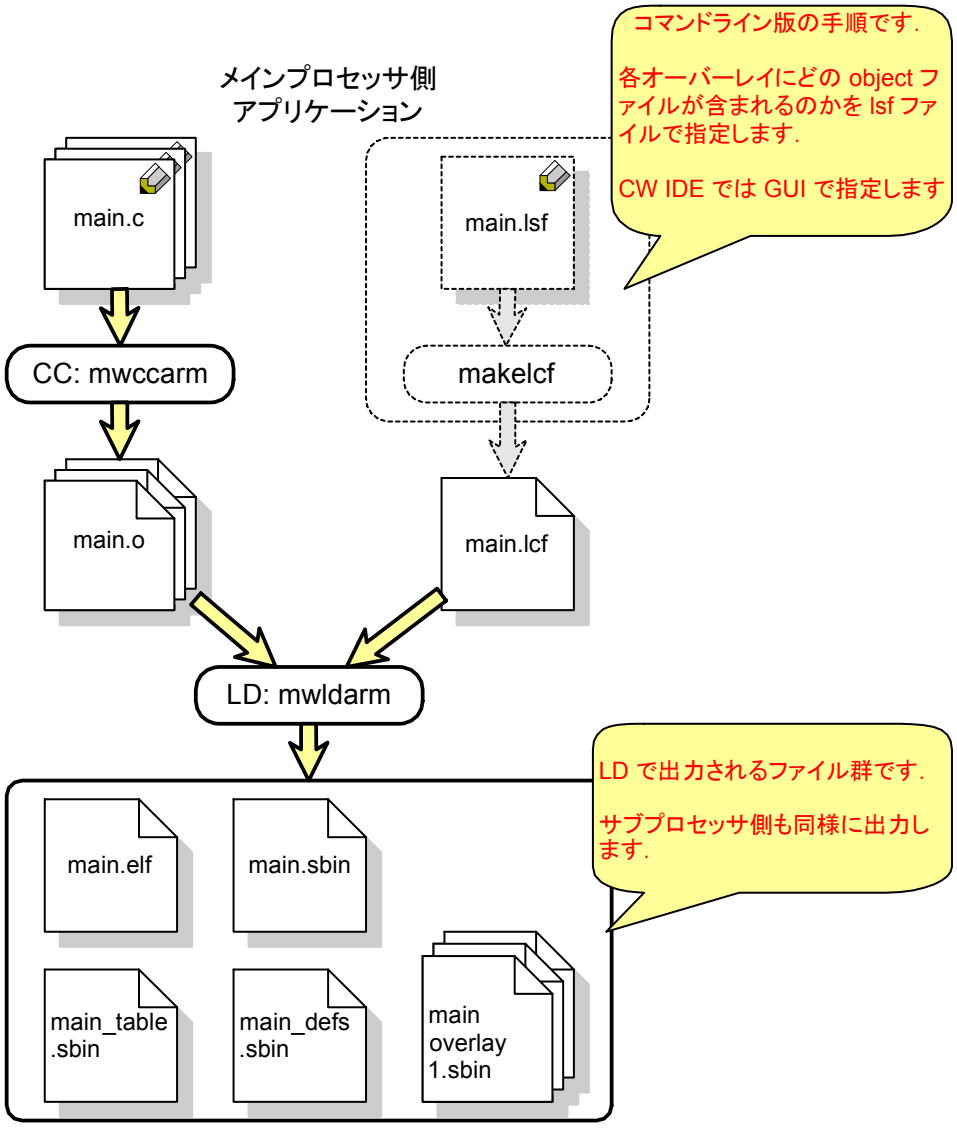
2. NitroROM 作成パス

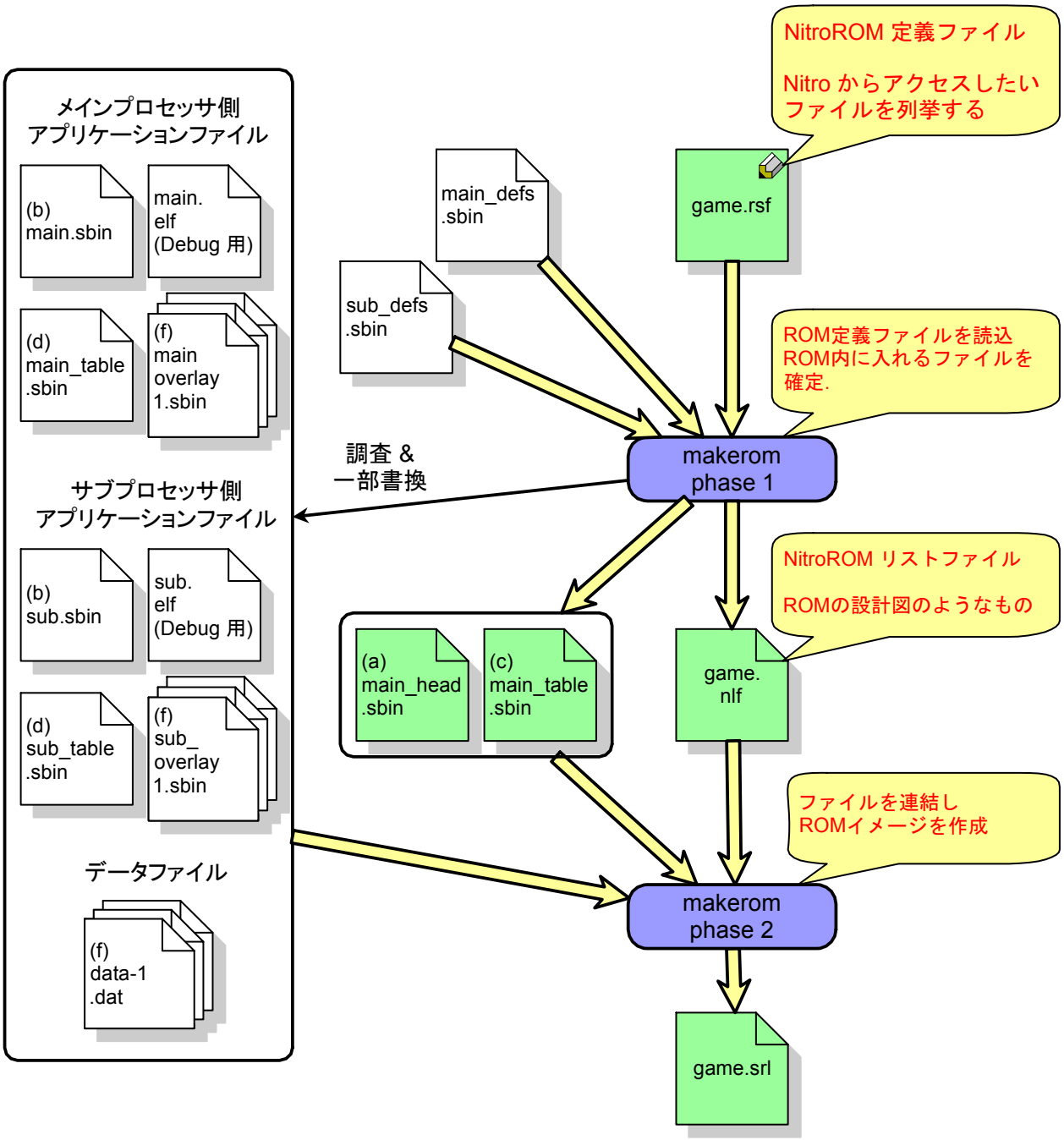
ROM ファイルは下図のようなパスで生成されます。
ROM ファイル作成には makerom と呼ばれるアプリケーションが重要な働きをします。 makerom は、それぞれ

フェーズ1
- ROM 内に入れるファイルを確定させ、前項の〝f) ファイルイメージブロック〟内での各ファイルのオフセット位置を決定しその情報をファイルとして出力します。
- またこのとき同時に〝c)ファイル名テーブル〟の完全な形と〝a) ROM ヘッダ〟の雛形を生成します。
- さらに〝d)オーバーレイヘッダテーブル〟に情報を追加し完全な形にします。

フェーズ2
- フェーズ1で出力した情報ファイルを元にして〝e)ファイルアロケーションテーブル〟と〝f）ファイルイメージブロック〟を生成します。
- 〝a) ROM ヘッダ〟に情報を追加し完全な形にします。
- a）～ f）のすべてを連結し、ROM ファイルとします。

という処理を行ないます。


メインプロセッサ側
アプリケーション
main.c
main.lsf
コマンドライン版の手順です.
各オーバーレイにどの object ファイルが含まれるのかを lsf ファイルで指定します.
CW IDE では GUI で指定します
CC: mwccarm
makelcf
main.o
main.lcf
LD: mwldarm
LD で出力されるファイル群です.
サブプロセッサ側も同様に出力します.
main.elf
main.sbin
main_table
.sbin
main_defs
.sbin
main
overlay
1.sbin



`makerom` では以下の設定ファイルを使用します。

a)　NitroROM スペックファイル　　拡張子 `.rsf`
b)　NitroROM リストファイル　　拡張子 `.nlf`

これらについて説明します。

a) NitroROM スペックファイル .rsf

定義ファイルの例です。詳しいフォーマットについては makerom ツールのリファレンスマニュアルを参照してください。

リンカが出力した部分バイナリファイル(.sbin)を Arm9/Arm7 セクションにそれぞれ記載し、ROM ファイル内に追加したいファイルを RomSpec セクションに追加します。またその他の情報を Property セクションに記述します。

以下の例では、開発 PC の data/graphics ディレクトリ内の拡張子.jpg のファイルを ROM ファイルの /data ディレクトリ以下に追加し、その次の領域には data/ARM7/sound 下の拡張子.wav のファイルを /sound ディレクトリ以下に追加しています。

```
#
#  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 リストファイル .nlf

ROMイメージを構築するための情報一式が含まれている CSV 形式のファイルです。rsf ファイルにおけるあいまいな部分が確定されています。以下は具体例です。

```
#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.sbin","main_ovt.sbin","main_ovn.sbin","."
7,"sub.nef","sub.sbin","sub_ovt.sbin","sub_ovn.sbin","."

# File Image Block
F,00000000,000201fc,00,001c,ffff,"ROMROOT/Nitro.ROM","/Nitro.ROM",3ffbf36e,512,0
P,000201fc,00000200,00
```

改行コードは "¥r¥n" です。
各行は、第一パラメータに1個の ASCII 文字からなるコマンドと、それ以降のパラメータから構成されます。
コマンドは付帯情報を示すコマンドであるヘッダコマンドと実際のROMの内容を規定するボディコマンドの2種類に分けることができます。ボディコマンドはヘッダコマンドの前に位置してはいけません。

[ヘッダコマンド]
Version:　[V]

　　V,[ファイルのバージョン番号]

　　ROM ファイルのフォーマットのバージョン番号です。
　　arm9files/arm7files に[オーバーレイバイナリのトップディレクトリからの相対パス] 情報が追加されたことにより 1.1 に変更されました。

Topdir:　[T]

　　T,[トップディレクトリ名]

　　ファイルの間接パス指定表現における基準ディレクトリです。
　　NLF ファイルが生成された時のカレントディレクトリが設定されます。
　　トップディレクトリが相対パスで指定されている場合、トップディレクトリの位置は NLF ファイルが置かれているディレクトリを基準として解釈されます。

Headers:　　[H]

　　H,[ROMヘッダファイル],[ファイルネームテーブルファイル],[ファイルIDを持つファイルの数]

　　ROM ヘッダファイルおよびファイルネームテーブルのファイル名を指定します。
　　これらのファイルは makerom で作成されます。
　　ファイル名はダブルクォーテーションで囲まれています。
　　[ファイルIDを持つファイルの数] を8倍した値がファイルアロケーションテーブルのサイズになります。

arm9files:　[9]

　　9,[nef ファイル],[常駐モジュール bin ファイル],
　　　　[オーバーレイテーブルファイル],
　　　　[オーバーレイネームファイル],[オーバーレイバイナリのトップディレクトリからの相対パス]
　　（見やすいように改行を入れていますが、実際は一行です）

　　メインプロセッサ用アプリケーションファイルのファイル名を指定します。
　　各ファイル名はダブルクォーテーションで囲まれています。
　　オーバーレイネームテーブル内のバイナリへのアクセスパスは［トップディレクトリ］と
　　[オーバーレイバイナリのトップディレクトリからの相対パス] を連結することで取得できます。
　　オーバーレイテーブルファイルが必要ないときはファイル名として ″*″ を設定します。このときは[オーバー
　　レイバイナリのトップディレクトリからの相対パス] にも ″*″ が設定されます。

arm7files:　[7]

　　7,[nef ファイル],[常駐モジュール bin ファイル],
　　　　[オーバーレイテーブルファイル],
　　　　[オーバーレイネームファイル],[オーバーレイバイナリのトップディレクトリからの相対パス]
　　（見やすいように改行を入れていますが、実際は一行です）

　　サブプロセッサ用アプリケーションファイルのファイル名を指定します。
　　各ファイル名はダブルクォーテーションで囲まれています。
　　オーバーレイテーブルファイルが必要ないときはファイル名として ″*″ を設定します。このときは[オーバー
　　レイバイナリのトップディレクトリからの相対パス] にも ″*″ が設定されます。

[ボディコマンド]
File:　　[F]

　　F,[開始オフセット],[終了オフセット],[パディングコード],
　　　　[ファイルID],[ロムヘッダオフセット],
　　　　[開発機側ファイル名],[ROMファイル名],[ファイルのタイムスタンプ],

[アラインメント値],[移動禁止フラグ 0:移動可 1:禁止]
（見やすいように改行を入れていますが、実際は一行です）

ファイルを ROM ファイルとしてファイルイメージブロックへ追加します。
対象となるファイルが置かれるROMのアドレス空間は下記の address の範囲となります。

開始オフセット ≦ address ＜ 終了オフセット

ファイルが指定されたアドレス空間よりも小さく、開始オフセットと終了オフセットで指定した範囲を埋めることが出来ない場合、残りの領域はパディングコードで埋められます。逆にファイルの方が大きい場合はエラーとなります。

ファイルのROM内でのオフセット位置に関する情報を、[ファイルID] の指定値に従ってファイルアロケーションテーブルに格納します。格納する位置はファイルアロケーションテーブルの先頭から [ファイルID]*8 バイトの位置となります。格納構造は前章のファイルアロケーションテーブルの説明をご覧ください。また、指定値が ffff になっている場合はそのオフセット情報をファイルアロケーションテーブルに格納する必要はありません。
同様に、[ロムヘッダオフセット] が ffff でないファイルはロムヘッダの所定の位置にファイルのオフセット情報を格納します。格納する位置はロムヘッダの先頭から [ロムヘッダオフセット] *4 バイトの位置となります。詳しくはロムヘッダの説明をご覧ください。

[開発機側ファイル名] は開発PC上におけるファイルの位置を示しています。
ファイルアロケーションテーブルは makerom 時に生成されるため PC 上のファイルとして存在しません。このファイルアロケーションテーブルの挿入位置を指定するために特別な名前のファイル名 *FILEALLOC が定義されています。

通常のファイル以外、例えばオーバーレイテーブルのようにファイルシステム構築に関するデータファイルには ROM ファイル名が設定されていません。このような名前の無いファイルの [ROM ファイル名] 欄にはアスタリスク（＊）が指定されています。これは sscanf によるコマンド文字列の解析を容易にするための措置です。

[ファイルのタイムスタンプ]は C ライブラリの time_t 型の値(1970年[UTC] からの通算秒数)であり、stat 関数で取得される stat 構造体のメンバー st_mtime の値となります。ファイルアロケーションテーブル(*FILEALLOC)の場合、このフィールドには 0 が設定されています。

[移動禁止フラグ] が 1 であるファイルについては、開始オフセット/終了オフセットの値を変更することが許可されていません。
このフラグが 0 である場合はオフセットの変更が許可されています。このときは、アラインメント値で指定された数値単位で各オフセットを変更するようにしてください。

各値のフォーマットは下記の通りです。(sscanf の表記)

コマンド	%1c
開始オフセット	%08x
終了オフセット	%08x
パディングコード	%02x

ファイルID	%04x
ロムヘッダオフセット	%04x
開発機側ファイル名	¥"%1024[^¥"¥n]¥"
ROMファイル名	¥"%1024[^¥"¥n]¥"
ファイルのタイムスタンプ	%08x
アラインメント値	%d
移動禁止フラグ	0 または 1

Padding:　[P]

P,[開始オフセット],[終了オフセット],[パディングコード]

パディングを ROM ファイルイメージブロックへ追加します。
対象となるROMのアドレス空間は下記の address の範囲となります。

開始オフセット ≦ address ＜ 終了オフセット

開始オフセットと終了オフセットで指定した範囲がパディングコードで埋められます。

[その他]
#comment:　[#]

[コメント]

何もしません。コメント用です。この行に関してはカンマ区切りは必要ありません。

#NLF
ファイルのマジックナンバーとしての用途を考え、ファイルの先頭の 4 文字は '#NLF' となっています。

3 オーバーレイ処理

オーバーレイパラメータを makerom 実行時に取得するために、リンク時に以下の形式のオーバーレイネームファイルを作成します。

これには常駐モジュール関連の起動時のパラメータ(16バイト)と、リンク時に生成された各オーバーレイの実行バイナリのファイル名が保存されています。

オーバーレイ実行バイナリファイル名はオーバーレイIDの順に'¥0'文字で終了する文字列として、パックされています。例えばファイル名が"a.sbin","b.sbin","c.sbin" であったときにはファイル名データは

"a.sbin¥0b.sbin¥0c.sbin¥0"

という形式で保存されます。オーバーレイ ID が N であるオーバーレイのファイル名を取得するには、ファイル名データの先頭から N 番目の'¥0'を検索し、その次の文字から始まる文字列をファイル名として取

得します。

```
//
// OverlayDefs フォーマット
//
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[];       // 可変長 STRING データ
                                        // NULL 文字で終了するファイル名が
                                        // オーバーレイバイナリに対応する数だけ保持されている
} ROM_ONT;
```