

組み込みマニュアル

ADX ファイルシステム

1998年 10月23日

Ver . 5.18

1999年 03月16日

Ver . 5.50

1999年 06月11日

Ver . 5.53

1999年 08月18日

Ver . 5.54

1999年 12月22日

Ver . 5.55

2000年 03月07日

Ver . 5.55

変 更 履 歴

年月日	バージョン	変 更 内 容
1998.10.23	5.18	新規作成。
1999.03.16	5.50	ADXF_ReadSj32 関数を追加。
1999.06.11	5.53	ライブラリのバージョンアップに伴いバージョン番号を更新。
1999.08.18	5.54	ハンドルの状態にエラー状態を追加。
1999.12.22	5.55	付録として、ドアオープンチェック、リードエラー、シーク音の軽減、データ読み込みの高速化の説明を追加。 記述ミスを修正。
2000.03.07	5.55	記述ミスを修正。

目 次

1. 概 要	1
1.1 目 的	1
1.2 特 徴	1
1.3 モジュール構成	1
2. 基本事項	2
2.1 用 語	2
3. ADX ファイルシステムの使用方法	3
3.1 ISO9660 ファイルの読み込み方	4
3.2 AFS ファイルへのアクセス方法	5
3.2.1 AFS ファイルの作成方法	5
3.2.2 インサイドファイルの読み込み方	6
3.2.3 追記したインサイドファイルの読み込み方	7
4. データ仕様	8
4.1 定 数	9
4.2 データ型	10
5. 関数仕様	11
5.1 ライブラリの初期化と終了	12
5.2 データの読み込み	15
5.3 アクセスポインタの制御	19
5.4 情報の取得	20
6. 付録	22
6.1 ドアオープンチェック	22
6.2 リードエラー	23
6.3 シーク音の軽減	24
6.3.1 ディレクトリチェンジ	24
6.3.2 AFS ファイルの有効利用	24
6.4 データ読み込みの高速化	25
6.5 ストリームジョイントによる読み込み	26

1. 概 要

1.1 目 的

本システムは、音声をストリーム再生させながら、データを読み込むことを目的としています。

1.2 特 徴

本システムの特徴を以下に示します。

- (1) 音声をストリーム再生させながら、データを読み込むことができます。
- (2) ISO9660 ファイル(普通のファイル)の他に、複数のファイルを結合したファイル(AFS ファイル)を読み込むことができます。
- (3) 複数のファイルを同時に開き、並列に読み込むことができます。

1.3 モジュール構成

ADX ファイルシステムのモジュール構成図を以下に示します。

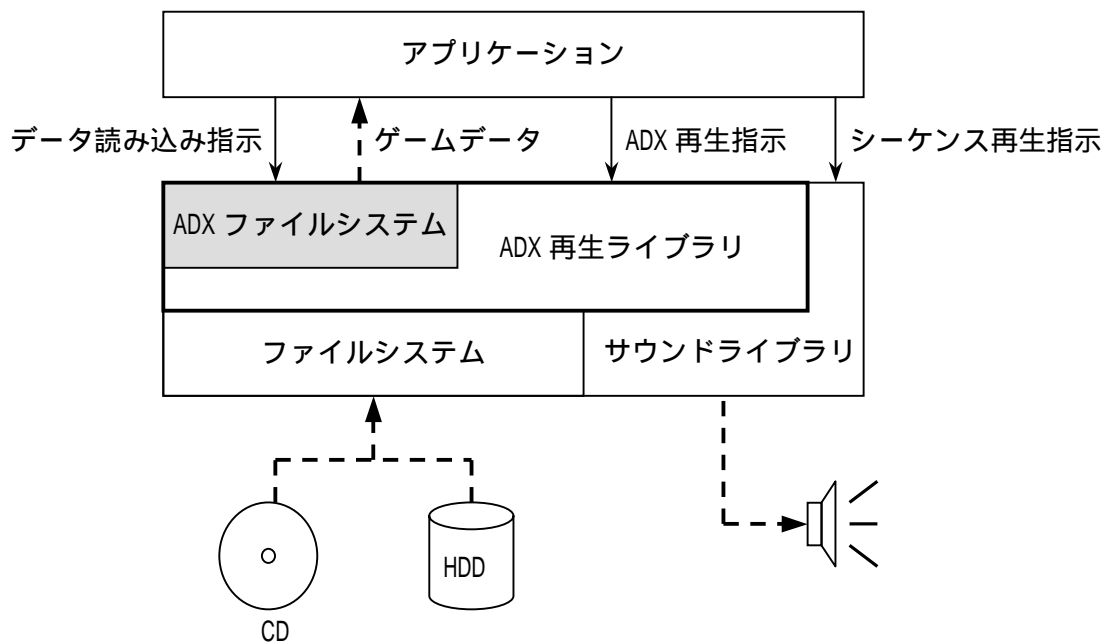


図 1 - 1 モジュール構成

2. 基本事項

2.1 用語

ADX ファイルシステムの説明で使用する用語を以下に示します。

表 2 -1 用語一覧

用語	説明
ISO9660 ファイル	ISO9660 フォーマットのファイル(いわゆる、普通のファイル)。
AFS ファイル	複数の ISO9660 ファイルを PC 上で結合したファイル。 最大 65536 個のファイルを結合することができます。
パーティション	AFS ファイル内のファイルをファイルとして扱い、AFS ファイルをディスクの区画として扱うための呼び名。 ライブラリによるディスクからの読み込む側の説明で使用しています。
インサイドファイル	AFS ファイル(パーティション)内に結合されているファイル。 サイズは最大 128Mbyte までで、2048byte(セクタ)単位となります。
パーティション ID	パーティションを識別するための識別子。
ファイル ID	インサイドファイルを識別するための識別子。

3. ADX ファイルシステムの使用方法

以下の手順により、ディスクからメモリへのデータ転送を行います。

- (1) ファイルをオープンします。 (ADXF_Open/ADXF_OpenAfs)
- (2) 読み込みリクエストを発行します。 (ADXF_ReadNw32)
- (3) 読み込み終了状態になるまで、状態を管理します。 (ADXF_GetStat)
- (4) 終了状態になったら、ファイルをクローズします。 (ADXF_Close)

ハンドルの状態を以下に示します。

表 3 - 1 ハンドルの状態

状 態	説 明
STOP	データ読み込みを停止している状態
READING	データ読み込み中の状態
READEND	データ読み込みが終了した状態
ERROR	エラー状態

ハンドルの状態は、ADXF_ReadNw32 関数により STOP から READING に変わり、指定のセクタ数のデータを読み込み終わると、自動的に READEND に変わります。

状態遷移図を以下に示します。

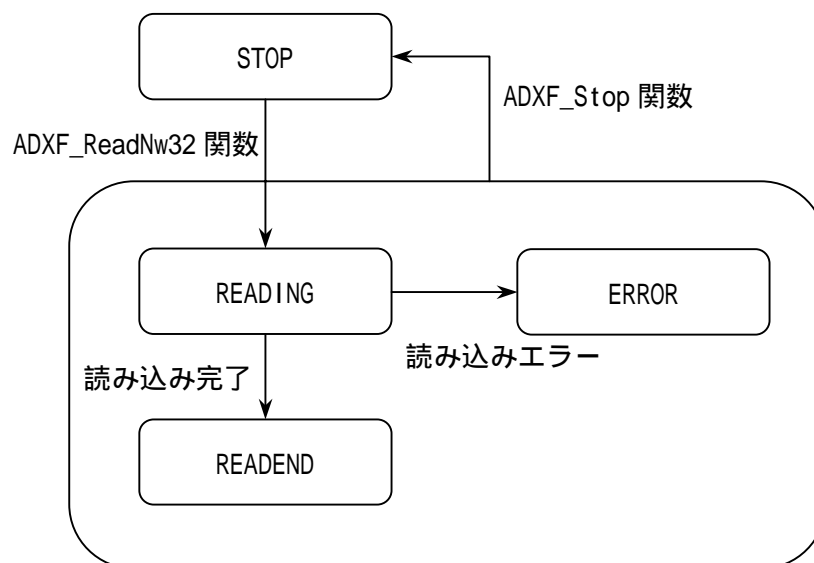


図 3 - 1 状態遷移図

3.1 ISO9660 ファイルの読み込み方

以下に、ISO9660 ファイルを読み込むサンプルプログラムを示します。

< サンプルプログラム >

/* アプリケーションメイン関数 */

```
void main(void)
{
    ADXF    adxf;                /* ADXF ハンドル          */
    long    rd_nsct = 1000;      /* 読み込みセクタ数      */
    char    *rd_buf = (char *)0x8C800000; /* バッファアドレス      */

    /*
     * H / Wなどの初期化
     */
    ADXT_Init();                /* ライブラリの初期化    */
    adxf = ADXF_Open("SAMPLE0.DAT"); /* ファイルオープン      */
    ADXF_ReadNw32(adxf, rd_nsct, rd_buf); /* 読み込み開始          */
    for (;;) {
        if (ADXF_GetStat(adxf) == ADXF_STAT_READEND) /* 読み込み終了チェック */
            break;
        /*
         * V-Sync 待ち
         */
    }
    ADXF_Close(adxf);           /* ファイルクローズ      */
}
```

3.2 AFS ファイルへのアクセス方法

AFS ファイルを使用することによって、大量のファイルを簡便に扱うことができます。複数のファイルをあらかじめ結合しておき、AFS ファイルとしてディスク上に格納します。ADX ファイルシステムは、結合された AFS ファイル内から、任意のインサイドファイルを読み込むことができます。

3.2.1 AFS ファイルの作成方法

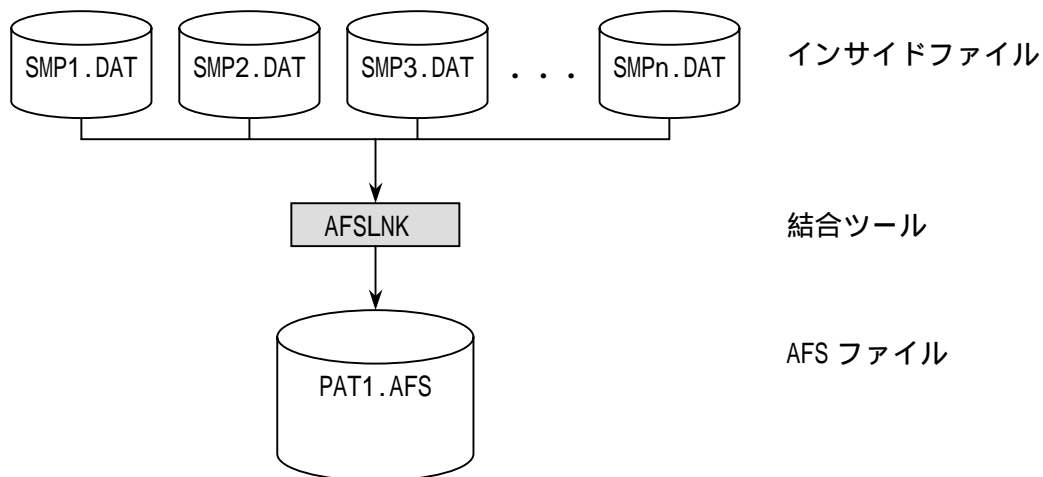


図 3-2 AFS ファイルの作成

AFS ファイルは、ADX データやグラフィックデータなどを連結したファイルです。afslnk.exe プログラムにより結合します。以下のようにファイルリストを引数として与えます。

C:¥TEMP>dir /b *.adx > pat01.als

カレントディレクトリ内の全ての ADX ファイルを

C:¥TEMP>afslnk pat01.als

結合した、pat01.afs が生成されます。

カレントディレクトリ以外のディレクトリに格納されているデータを結合する場合は、以下のよう
に指定します。

C:¥TEMP>dir /b pat01¥*.adx > pat01.als

pat01 ディレクトリ内の全ての ADX ファイル

C:¥TEMP>afslnk pat01.als -dir=pat01

を結合した、pat01.afs が生成されます。

3.2.2 インサイドファイルの読み込み方

以下では、インサイドファイルの読み込み方について説明するため、AFS ファイルをパーティションと呼びます。

インサイドファイルへのアクセスは、以下の手順により行います。

(1) パーティション情報の読み込み

ADXF_LoadPartition 関数により、パーティション情報を読み込みます。パーティション情報は、1つのインサイドファイル当たり約2byteの領域が必要です。

```
static char ptinfo[ADXF_CALC_PTINFO_SIZE(5)]; /* パーティション情報領域 */
long      pi_sz; /* パーティション情報サイズ */

ADXT_Init(); /* ライブラリの初期化 */
ADXF_LoadPartition(0, "PAT0.AFS", ptinfo, 5); /* パーティションのロード */
```

複数のパーティションを扱う場合は以下のように行います。1つのパーティション情報領域を複数のパーティションで使うことができます。

```
static char ptinfo[ADXF_CALC_PTINFO_SIZE(256)]; /* パーティション情報領域 */
long      pi_sz; /* パーティション情報サイズ */

ADXT_Init(); /* ライブラリの初期化 */
ADXF_LoadPartition(0, "PAT0.AFS", ptinfo, 5); /* パーティションのロード */
pi_sz = ADXF_GetPtinfoSize(0); /* パーティション情報サイズの取得 */
ADXF_LoadPartition(1, "PAT1.AFS", ptinfo+pi_sz, 8);
:
```

(2) パーティション内のデータの読み込み

ADXF_OpenAfs 関数により、ファイルをオープンし、グラフィックのデータファイルなど読み出します。

```
ADXF      adxf; /* ADXF ハンドル */
long      pati_id = 1; /* パーティション ID */
long      file_id = 3; /* ファイル ID */
long      rd_nsct = 1000; /* 読み込みセクタ数 */
char      *rd_buf = (char *)0x8C800000; /* バッファアドレス */

adxf = ADXF_OpenAfs(pati_id, file_id); /* ファイルオープン */
ADXF_ReadNw32(adxf, rd_nsct, rd_buf); /* 読み込み開始 */
for (;;) {
    if (ADXF_GetStat(adxf) == ADXF_STAT_READEND) /* 読み込み終了チェック */
        break;
    /* V-Sync 待ち */
}
ADXF_Close(adxf); /* ファイルクローズ */
```

3.2.3 追記したインサイドファイルの読み込み方

開発の途中段階で、インサイドファイルを追加・差し替えする場合、AFS ファイルを再結合すると作業効率が悪いので、追記機能を検討しています。現バージョンでは未実装のため説明を省きます。

4. データ仕様

ライブラリのデータ一覧を以下に示す。

表 4 - 1 データ一覧

データ名		機 能	番号
定 数			
ADXF_STAT_~		ハンドルの状態	1.1
ADXF_SEEK_~		シークタイプ	1.2
データ型			
ADXF		ADXF ハンドル	2.1

4.1 定 数

Title	Data Name	Data	No
データ	ADXF_STAT_ ~	ハンドルの状態	1.1

以下に示す定数は、ハンドルの状態を示す。

定数名	説 明
ADXF_STAT_STOP	データ読み込みを停止している状態
ADXF_STAT_READING	データ読み込み中
ADXF_STAT_READEND	データ読み込みが終了した状態
ADXF_STAT_ERROR	データ読み込みでエラーが発生した状態

Title	Data Name	Data	No
データ	ADXF_SEEK_ ~	シークタイプ	1.2

以下に示す定数は、アクセスポインタを移動する際の基準を示す。ADXF_Seek 関数の引数として使用する。

定数名	説 明
ADXF_SEEK_SET	ファイルの先頭
ADXF_SEEK_CUR	現在の位置
ADXF_SEEK_END	ファイルの終端

4.2 データ型

Title	Data Name	Data	No
データ	ADXF	ADXF ハンドル	2.1

ファイルアクセスに関する、ファイル毎の情報を保持する。ADXF_Open,ADXF_OpenAfs 関数によって生成される。ファイルアクセスのほとんどの関数がこのデータを参照する。

5. 関数仕様

ライブラリの関数一覧を以下に示す。

表 5 - 1 関数一覧

関数名	機 能	番号
ライブラリの初期化と終了		
ADXF_Init	ライブラリの初期化	1.1
ADXF_Finish	ライブラリの終了処理	1.2
ADXF_LoadPartition	パーティション情報の読み込み	1.3
ADXF_AddPartition	追記パーティション情報の読み込み	1.4
ADXF_GetPtinfoSize	パーティション情報サイズの取得	1.5
データの読み込み		
ADXF_Open	ファイルオープン (ISO9660 フォーマット)	2.1
ADXF_OpenAfs	ファイルオープン (AFS フォーマット)	2.2
ADXF_Close	ファイルクローズ	2.3
ADXF_ReadNw32	データの読み込み開始	2.4
ADXF_ReadSj32	ストリームジョイントへの読み込み開始	2.5
ADXF_Stop	データの読み込み停止	2.6
ADXF_ExecServer	サーバ関数	2.7
アクセスポインタの制御		
ADXF_Seek	アクセスポインタの移動	3.1
ADXF_Tell	アクセスポインタの取得	3.2
情報の取得		
ADXF_GetFsizeSct	ファイルサイズの取得	4.1
ADXF_GetNumReqSct	読み込み要求情報の取得	4.2
ADXF_GetNumReadSct	実際に読み込んだセクタ数の取得	4.3
ADXF_GetStat	ハンドル状態の取得	4.4

5.1 ライブラリの初期化と終了

ライブラリの初期化と終了のための関数である。

Title	Function Name	Function	No
関 数	ADXF_Init	ライブラリの初期化	1.1

[書 式] void ADXF_Init(void);

[入 力] なし

[出 力] なし

[関数値] なし

[機 能] ライブラリを初期化する。

[備 考] ADXF_Init 関数(ADX 再生ライブラリの初期化)から呼び出されるため、通常アプリケーション側から呼び出す必要はない。

Title	Function Name	Function	No
関 数	ADXF_Finish	ライブラリの終了処理	1.2

[書 式] void ADXF_Finish(void);

[入 力] なし

[出 力] なし

[関数値] なし

[機 能] ライブラリの終了処理をする。

Title 関 数	Function Name	Function	No
	ADXF_LoadPartition	パーティション情報の読み込み	1.3

[書 式] long ADXF_LoadPartition(long ptid, char *fname, void *ptinfo, long nfile);

[入 力] ptid : パーティション ID(0~255)

fname : AFS ファイルの名前

ptinfo : パーティション情報読み込み領域

nfile : ファイル数

[出 力] なし

[関数値] エラーコード

[機 能] AFS ファイルのパーティション情報を読み込み、パーティション ID に設定する。

Title 関 数	Function Name	Function	No
	ADXF_AddPartition	追記パーティション情報の読み込み	1.4

[書 式] long ADXF_AddPartition(long ptid, char *fname, void *ptinfo, long nfile);

[入 力] ptid : パーティション ID(0~255)

fname : AFS ファイルの名前

ptinfo : パーティション情報読み込み領域

nfile : ファイル数

[出 力] なし

[関数値] エラーコード

[機 能] AFS ファイルのパーティション情報を読み込み、追記情報としてパーティション ID に設定する。

Title 関 数	Function Name ADXF_GetPtinfoSize	Function パーティション情報サイズの取得	No 1.5
--------------	-------------------------------------	-----------------------------	-----------

[書 式] long ADXF_GetPtinfoSize(long ptid);

[入 力] ptid: パーティション ID

[出 力] なし

[関数値] パーティション情報のサイズ(単位: バイト)

[機 能] 設定したパーティション情報のサイズを取得する。

5.2 データの読み込み

データの読み込み関数である。

Title	Function Name	Function	No
関 数	ADXF_Open	ファイルオープン (ISO9660 フォーマット)	2.1

[書 式] ADXF ADXF_Open(char *fname, void *atr);

[入 力] fname : ファイル名

atr : ファイルの属性

[出 力] なし

[関数値] ADXF ハンドル、エラーの場合は NULL

[機 能] 指定されたファイルをオープンし、ADXF ハンドルを返す。

Title	Function Name	Function	No
関 数	ADXF_OpenAfs	ファイルのオープン (AFS フォーマット)	2.2

[書 式] ADXF ADXF_OpenAfs(long ptid, long flid);

[入 力] ptid : パーティション ID

flid : ファイル ID

[出 力] なし

[関数値] ADXF ハンドル、エラーの場合は NULL

[機 能] パーティション ID とファイル ID で指定された AFS ファイルをオープンし、
ADXF ハンドルを返す。

Title 関 数	Function Name	Function	No
	ADXF_Close	ファイルのクローズ	2.3

[書 式] void ADXF_Close(ADXF adxf);
[入 力] adxf : ADXF ハンドル
[出 力] なし
[関数値] なし
[機 能] 指定された ADXF ハンドルをクローズする。

Title 関 数	Function Name	Function	No
	ADXF_ReadNw32	データの読み込み開始	2.4

[書 式] long ADXF_ReadNw32(ADXF adxf, long nsct, void *buf);
[入 力] adxf : ADXF ハンドル
 nsct : 読み込みデータ量(単位: セクタ)
 buf : 読み込み領域
[出 力] なし
[関数値] 読み込みデータ量(単位: セクタ)
[機 能] データ読み込みのリクエストを発行する。
 リクエストしたアクセス動作が完了すると、アクセスポインタは nsct セクタ進む。
[備 考] buf のアドレス境界は 32 バイトにする必要がある。
 ファイル末をまたぐような読み込みセクタ数を指定された時は、ファイル末までの
 セクタ数を返し、ファイル末までの読み込み処理を行う。

Title 関 数	Function Name	Function	No
	ADXF_ReadSj32	ストリームジョイントへの読み込み開始	2.5

[書 式] long ADXF_ReadSj32(ADXF adxf, long nsct, SJ sj);

[入 力] adxf : ADXF ハンドル
nsct : 読み込みデータ量(単位: セクタ)
sj : ストリームジョイント

[出 力] なし

[関数値] 読み込みデータ量(単位: セクタ)

[機 能] ストリームジョイントにデータ読み込みのリクエストを発行する。
ストリームジョイントのバッファサイズは、セクタの整数倍でなければならない。
ストリームジョイントからユーザがデータを読み出すと、自動的にストリームジョイントにデータが読み込まれる。
リクエストしたアクセス動作が完了すると、アクセスポインタは nsct セクタ進む。

Title 関 数	Function Name	Function	No
	ADXF_Stop	データの読み込み停止	2.6

[書 式] long ADXF_Stop(ADXF adxf);

[入 力] adxf : ADXF ハンドル

[出 力] なし

[関数値] 中止した時点でのアクセスポインタの位置

[機 能] データの読み込みを停止する。

Title 関 数	Function Name ADXF_ExecServer	Function サーバ関数	No 2.7
--------------	----------------------------------	-------------------	-----------

[書 式] void ADXF_ExecServer(void);

[入 力] なし

[出 力] なし

[関数値] なし

[機 能] 内部状態を更新する。

[備 考] V-Sync 割込みで自動的に呼び出されるため、通常アプリケーション側から呼び出す必要はない。

5.3 アクセスポインタの制御

アクセスポインタに制御する関数である。

Title 関 数	Function Name	Function	No
	ADXF_Seek	アクセスポインタの移動	3.1

[書 式] long ADXF_Seek(ADXF adxf, long pos, long type);

[入 力] adxf : ADXF ハンドル

pos : アクセスポインタ移動量(単位:セクタ)

type : 移動基準(シークタイプ: ADXF_SEEK_~)

[出 力] なし

[関数値] アクセスポインタの位置

[機 能] アクセスポインタを type から pos セクタ離れた位置に移動させる。

ファイル末尾を越える位置への移動が指定された場合は、ファイルの末尾にアクセスポインタを移動する。

定数名	説 明
ADXF_SEEK_SET	ファイルの先頭
ADXF_SEEK_CUR	現在の位置
ADXF_SEEK_END	ファイルの終端

Title 関 数	Function Name	Function	No
	ADXF_Tell	アクセスポインタの取得	3.2

[書 式] long ADXF_Tell(ADXF adxf);

[入 力] adxf : ADXF ハンドル

[出 力] なし

[関数値] アクセスポインタの位置(単位:セクタ)

[機 能] アクセスポインタの位置を取得する。

5.4 情報の取得

Title	Function Name	Function	No
関 数	ADXF_GetFsizeSct	ファイルサイズの取得	4.1

[書 式] long ADXF_GetFsizeSct(ADXF adxf);
[入 力] adxf : ADXF ハンドル
[出 力] なし
[関数値] ファイルサイズ(単位: セクタ)
[機 能] 指定されたファイルのサイズを取得する。

Title	Function Name	Function	No
関 数	ADXF_GetNumReqSct	読み込み要求情報の取得	4.2

[書 式] long ADXF_GetNumReqSct(ADXF adxf, long *seekpos);
[入 力] adxf : ADXF ハンドル
[出 力] seekpos : 読み込み位置
[関数値] 要求した読み込みデータ量(単位: セクタ)
[機 能] ADXF_ReadNw32 関数で要求した読み込み位置とデータ量を取得する。

Title 関 数	Function Name ADXF_GetNumReadSct	Function 実際に読み込んだセクタ数の取得	No 4.3
--------------	-------------------------------------	-----------------------------	-----------

[書 式] long ADXF_GetNumReadSct(ADXF adxf);

[入 力] adxf : ADXF ハンドル

[出 力] なし

[関数値] 読み込んだデータ量(単位:セクタ)

[機 能] 実際に読み込んだデータ量を取得する。

Title 関 数	Function Name ADXF_GetStat	Function ハンドルの状態の取得	No 4.4
--------------	-------------------------------	------------------------	-----------

[書 式] long ADXF_GetStat(ADXF adxf);

[入 力] adxf : ADXF ハンドル

[出 力] なし

[関数値] ADXF ハンドルの内部状態(状態:ADXF_STAT_~)

[機 能] ADXF ハンドルの内部状態を取得する。

定数名	説 明
ADXF_STAT_STOP	データ読み込みを停止している状態
ADXF_STAT_READING	データ読み込み中
ADXF_STAT_READEND	データ読み込みが終了した状態
ADXF_STAT_ERROR	データ読み込みでエラーが発生した状態

6. 付録

6.1 ドアオープンチェック

以下に、ドアオープンチェックのサンプルプログラムを示します。

< サンプルプログラム >

```
/* GD ファイルシステムのエラー発生時に起動する関数 */
void UsrGdErrFunc(void *obj, Sint32 errcode)
{
    if (errcode == GDD_ERR_TRAYOPEND || errcode == GDD_ERR_UNITATTENT) {
        ADXF_Finish();                                /* ADXF の終了処理 */
        sbExitSystem();                                /* Shinobi ライブラリの終了処理 */
        syBtExit();                                    /* シンプルプレイヤヘジャンプ */
    }
}

/* ユーザが V-SYNC 割り込みに登録する関数 */
void UsrVsyncFunc(void)
{
    Sint32 dstat;

    /* ドアオープンチェック */
    dstat = gdFsGetDrvStat();
    if (dstat == GDD_DRVSTAT_OPEN || dstat == GDD_DRVSTAT_BUSY) {
        gdFsReqDrvStat();
    }
}

/* アプリケーションメイン */
void main(void)
{
    :
    :
    /* GD ファイルシステムエラーコールバック関数の登録 */
    gdFsEntryErrFuncAll((void *)UsrGdErrFunc, NULL);
    njSetVSyncFunction(UsrVsyncFunc);
    :
    :
}
```

6.2 リードエラー

ADX ファイルシステムでは、リードエラーが発生すると、ハンドルの状態がエラー状態へ遷移します。従って、ハンドル状態を監視して、リードエラー発生時はアプリケーション側で対処して下さい。

```
ADXF_ReadNw32(adxf, nsct, buf);  
for (;;) {  
    if (ADXF_GetStat(adxf) == ADXF_STAT_ERROR) {  
        /* リードエラー処理 */  
    }  
}
```

6.3 シーク音の軽減

6.3.1 ディレクトリチェンジ

ディレクトリチェンジを行うと、シークが発生してデータ読み込みが1秒以上遅くなります。通常、ゲームデータは最外周から配置され、ディレクトリ情報は最内周に存在します。そのため、ディレクトリチェンジを行うと、ディレクトリ情報取得のために最内周までシークします。最外周から最内周までシークするのに約600[msec]程度かかるため、往復で1秒以上データ読み込みが遅くなります。

AFS ファイルを利用して、使用するデータを近くに配置することで、データをディレクトリと同様に管理しながら、シーク音を軽減することができます。

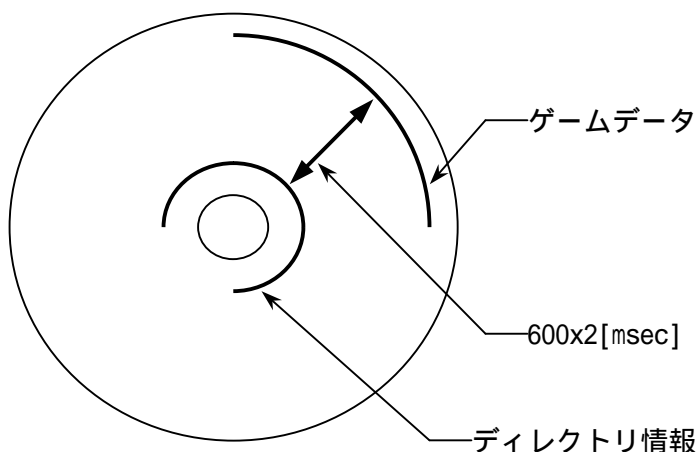


図 6 - 1 ディレクトリチェンジ

6.3.2 AFS ファイルの有効利用

BGM データ、セリフデータ、...とそれぞれに AFS ファイルを作成せず、各シーン毎に AFS ファイルを作成して下さい。マルチストリームで BGM を鳴らしながら、セリフを再生したり、グラフィックデータを読み込むような場合、データの種類ごとに AFS ファイルを作成すると、シーク時間が長くなります。

各シーン毎に AFS ファイルを作成することにより、マルチストリーミング時のシークを軽減することができます。

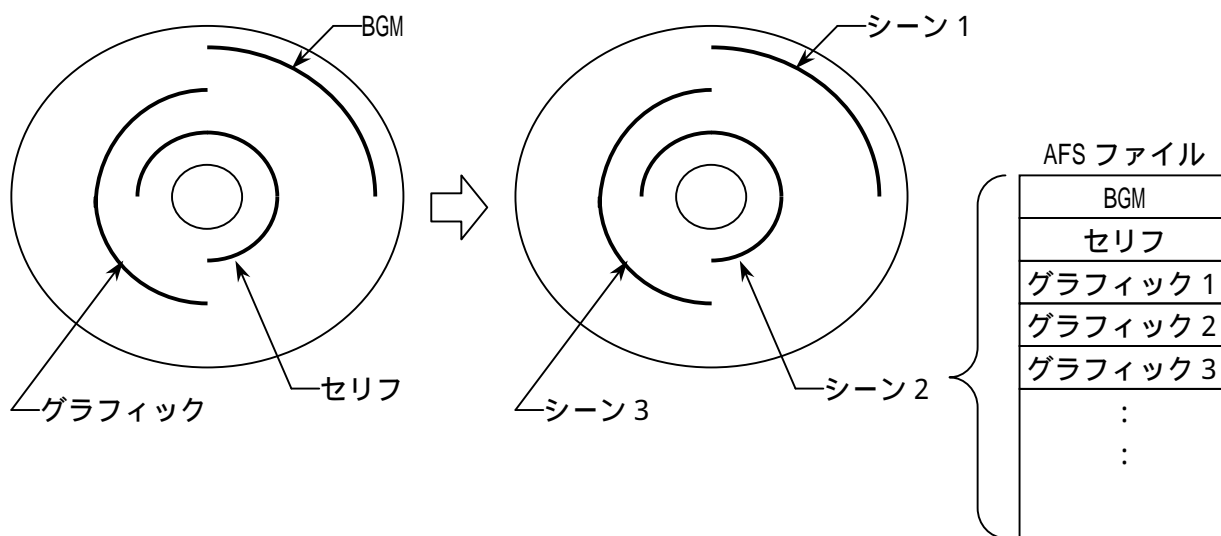


図 6 - 2 AFS ファイルの有効利用

6.4 データ読み込みの高速化

ADX ファイルシステムにより、GD-ROM から音声を再生している最中に、ゲームデータを読み込むことができます。しかしながら、デフォルトの設定では、ADX 再生の入力バッファの 85%を下回るとデータの再読み込みが始まるため、複数のゲームデータを読み込む場合、ゲームデータと音声データを交互に読み込むことになります。ADXT_SetReloadSct 関数を使用し、音声データの再読み込み開始量を調整することにより、ゲームデータの読み込みを高速化できます。入力バッファに最低 1 秒分のデータを読み込んでおくことで、音声を滞りなく再生できるので、再読み込み開始量は音声データの 1 秒分を指定します。

例えば、44.1KHz のステレオ音声は約 50Kbyte/sec なので、25 セクタ(1 セクタ=2048 バイト)を指定することになります。また、入力バッファを大きくすることにより、再読み込みの間隔を大きくすることができるので、更にゲームデータ読み込みを高速化できます。暫定仕様なので、ADXT_SetReloadSct 関数はリファレンスマニュアルには記載されていません。将来、このような指定をしなくても高速にゲームデータが読み込めるようになります。

```
/* 44KHz のステレオデータを再生する場合 */
#define WKSIZE      ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

char work[WKSIZE];          /* 作業領域 */
ADXT adxt;                  /* ADXT ハンドル */
ADXF adxf;                  /* ADXF ハンドル */

adxt = ADXT_Create(2, work, WKSIZE); /* ADXT ハンドルの生成 */
ADXT_SetReloadSct(adxt, 25); /* 再読み込み量の設定 */
:
ADXT_StartFname(adxt, "BGM.ADX"); /* 音声再生開始 */
:
adxf = ADXF_Open("GAMEDAT1.BIN", NULL); /* ファイルオープン */
ADXF_ReadNw32(adxf, nsct, buf); /* データ読み込み */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND );
ADXF_Close(adxf); /* ファイルクローズ */
:
(GAMEDAT1 が 2 秒以内に読み込めれば音声データの再読み込みは発生しない)
:
adxf = ADXF_Open("GAMEDAT2.BIN", NULL); /* ファイルオープン */
ADXF_ReadNw32(adxf, nsct, buf); /* データ読み込み */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND );
ADXF_Close(adxf); /* ファイルクローズ */
```

最大 GD-ROM ストリーム数(ADXT_CALC_WORK の第 3 引数)を増やすことにより、入力バッファを増やすことができます。引数の値を 1 増やすごとに 1 秒分の余裕ができます。

また、BGM とセリフを同時に再生するが、データを読み込むときにはセリフが再生されないことが保証されていれば、セリフのためのバッファ量をデータ読み込みに割り当てることができます。

6.5 ストリームジョイントによる読み込み

ストリームジョイントによるデータ読み込みを行うことで、アプリケーションは容易にストリーミングすることができます。ADX ファイルシステムは、ストリームジョイントに空き領域があると、自動的にデータを読み込みます。

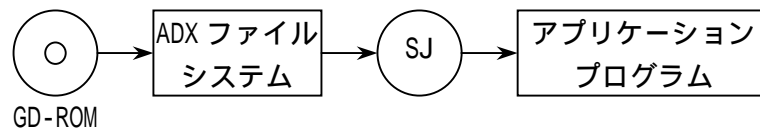


図 6-3 ストリームジョイントによる読み込み

この機能により、例えば Sofdec ファイルを AFS ファイルのインサイドファイルとして読み込むことができます。

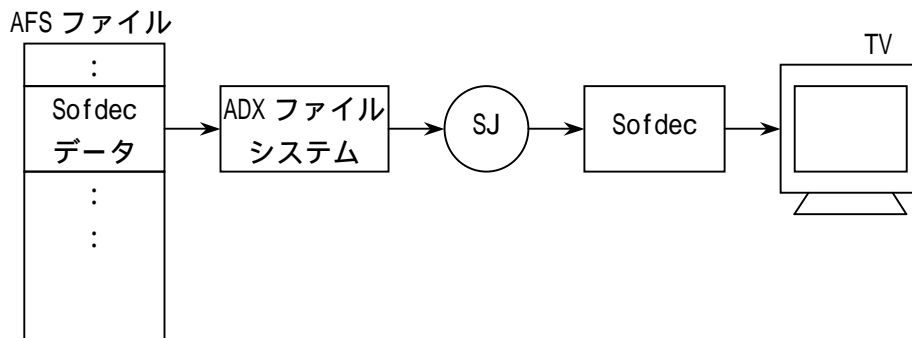


図 6-4 ストリームジョイントによる Sofdec の再生

< サンプルプログラム >

```
adxf = ADXF_OpenAfs(PAT0, FID1);
sj = mwPlyGetInputSj(sj);
ADXF_ReadSj32(adxf, fsize, sj);
:
```

ストリームジョイントを使用して、可変ブロック長の圧縮データを読み込み展開するサンプルプログラムを以下に示します。

< サンプルプログラム >

```
char    buf[64*1024+2048];           // 最大ブロック長が 2048byte

sj = SJRBF_Create(buf, 64*1024, 2048); // 最大ブロック長分エキストラ領域として指定。
                                         // この指定により 2048 バイトの連続を保証。

ADXF_ReadSj32(adxf, 1000, sj);
for (;;) {
    SJ_GetChunk(sj, SJ_LIN_DATA, 2048, &ck);
    nused = user_decode(ck.data, ck.len); // ユーザデコーダ
    SJ_SplitChunk(&ck, nused, &ck, &ck2);
    SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    SJ_UngetChunk(sj, SJ_LIN_DATA, &ck2);
}
```