開発を効率化するためファイルシステム

– アーカイブファイルを効果的に扱うファイルマネージャ –

Draft

2014 年 2 月 18 日　初稿

板垣 衛

■ 改訂履歴

稿	改訂日	改訂者	改訂内容
初稿	2014 年 2 月 18 日	板垣 衛	（初稿）

Draft

■ 目次

Draft

■ 概略

コンテンツ制作効率とファイル読み込み効率を最適にするためのファイルシステムを設計する。

アセットのアーカイブ（オーサリング）と実機上のファイルマネージャについて扱う。
「アーカイブ」「差分アーカイブ」「パッチ」「ファイルキャッシュ」「自動リロード」「ホストへのファイル出力」といった要件に対応する。

なお、このファイルシステムを活用した制作業務のイメージは、別紙の「効果的なランタイムアセット管理」にも示している。

■ 目的

本書は、ファイルマネージャとリソースマネージャを密接に連携させることによる、効率的な基本ゲームシステムを構築することを目的とする。
それにより、汎用的で安全なリソースの自動リロードの仕組みを確立し、ランタイム中での手軽なトライ＆エラーを可能にし、快適な開発環境を構築する。

■ 設計にあたって

優秀なミドルウェアや SDK のファイルシステムをベースに、ゲーム処理部分のレイヤーとして設計を行うことを考えていたが、差分オーサリング、パッチ、キャッシュ、自動リロードなどを統合的に扱うことを考えた場合、独自に実装したほうがなにかと都合が良く、メモリ効率も良いため、完全に独自の設計とする。

■ 要件定義

▼ 基本要件

本書が扱うシステムの基本要件は下記の通り。

- 基本的に、アーカイブファイルをマウントして扱うファイルシステムとする。
- デイリービルドにより全アセットをオーサリングしたアーカイブファイルをベースに、

Draft

制作スタッフの PC 上でローカルオーサリングした差分のアーカイブファイルを組み合わせて使用するものとする。

- 他の制作スタッフに差分アーカイブファイルを受け渡して、複数のアーカイブファイルを組み合わせて使用することを可能とする。
- 複数のアーカイブファイルに同じパスのファイルが含まれる場合、優先度の高いアーカイブファイル内のものが採用される。（パッチシステムと同じ仕組み）
- 「自動リロード」用にアーカイブしたファイルを用意することで、ゲームを再起動することなく簡単にリロードできるものとする。（リソースマネージャと連動する）
- 一度読み込んだファイルは不要になってもすぐには破棄せず、以後の読み込みの邪魔にならない限りキャッシュとして存続するものとする。
- ファイル読み込みはできる限り効率化し、同時に読み込むべきファイルは極力まとめて読み込む。
- 圧縮ファイルを扱い、キャッシュ効率を上げることができるものとする。
- ゲーム上でデータをエディットしてホスト（PC）上に出力したり、そのファイルをまた読み込んだりするといった例外的な要件にも対応可能とする。

▼ 要求仕様／要件定義

以下、本書が扱うシステムの要件を定義する。なお、要件として不確定の要求仕様も併記する。

- 「ファイルシステム」は「ファイルマネージャ」によって管理される。
- 「ファイル読み込み」「ファイルの存在チェック」「ファイルサイズの確認」に対しては、必ずファイルマネージャを通さなければならない。
  - 直接ファイル操作系の API を使用してはいけない。
  - それ以前に「直接ファイル操作が必要か？」をよく考えて使わなければならない。
    - 例えば、「リソース構築要求」は「リソースマネージャ」に出して扱う。リソースマネージャがその要求に応じてファイルマネージャにファイル読み込み要求を出すので、各処理系が個別にリソースファイルを読み込むようなことはしない。

● システム間の依存関係

リソースマネージャとファイルマネージャは密接に連携するが、その依存関係は、リソースマネージャ⇒ファイルマネージャの一方向である。

- ファイルマネージャはリソースマネージャに依存しない。

Draft

- 自動リロードの際は、ファイルマネージャがリソースマネージャに働きかける必要があるが、この実現にはオブザーバー（コールバック）を用いる。
  - リソースマネージャがファイルマネージャに対してオブザーバー（コールバック処理）を登録しておくと、自動リロードファイル読み込み時にリソースマネージャに通知される。
- リソースマネージャについては、別紙の「開発の効率化と安全性のためのリソース管理」を参照。

● アーカイブファイルに関する要件

ファイルマネージャは、アーカイブファイルをマウントして扱う。

- モデルデータやアニメーションデータ、テクスチャ、メニューデータ、ゲームデータなど、各種アセットはそれぞれの方法でゲーム用にオーサリングを行うが、最終的なオーサリングファイルは、それらをまとめた単純なアーカイブファイルである。
- アーカイブファイルをマウントすると、そのファイルをオープンした状態になり、アーカイブファイルのヘッダー部分をメモリに展開して、アーカイブ内のファイルにアクセスできるようになる。
  - アーカイブファイルのヘッダー部分では、アーカイブ内のファイルリストが管理される。
  - ファイルリストでは、「ファイルパスの CRC 値」「ファイル拡張子の CRC 値」「ファイルの先頭位置」「圧縮ファイルサイズ」「本来のファイルサイズ」という情報が管理される。（非圧縮ファイルの場合、圧縮ファイルサイズは０）
  - ファイルリストは CRC 順に並び、そのままバイナリサーチ可能な状態で格納される。
  - アーカイブファイルのフッター部にはファイルパスなどの文字列情報が記録されており、別途読み込んでファイルパスの表示に用いることができる。（基本的にデバッグ用）
- アーカイブファイルは、最大サイズを 4GB とする。
  - これにより、ヘッダー部のファイルリストで扱う「位置情報」「サイズ情報」を 4 バイトの情報量に抑える。
  - 4GB を超えるデータが扱うには、複数のアーカイブファイルを用いる。
- アーカイブファイルは複数マウントすることができる。
  - マウントしたアーカイブファイルには優先度がある。
  - ファイル読み込み要求時は、優先度の高いアーカイブ内のファイルから順に探す。
  - フルオーサリングファイルよりもローカルオーサリングファイルを高い優先度に設定してマウントする。
  - この仕組みにより、そのままパッチシステムとして扱う。
- アーカイブファイル内にはさらにアーカイブファイルを含むことができる。
  - アーカイブ内アーカイブファイルは、複数のファイルをひとまとめに読み込んで読み込みを効率化するために使用する。

Draft

- アーカイブ内アーカイブ内のファイルは、親のアーカイブのファイルリストにも登録され、アーカイブ内のファイルであることを気にせず、直接読み込みを指定することができる。
  - ・ そのようなファイルの「位置情報」は、アーカイブファイルの先頭を指し、「サイズ情報」は本来のファイルのサイズを示す。
- アーカイブ内アーカイブ内のファイルが読み込まれる時、自動的にそのアーカイブ内の全ファイルがひとまとめに読み込まれる。
- イベント再生で使用するような巨大なアーカイブの場合、ひとまとめに読み込むのではなく、一時的にマウントして使用する。
  - ・ このようなアーカイブの場合は、親のアーカイブのファイルリストにファイルを展開しない。
  - ・ アーカイブをどう扱うかは、アーカイブ自体が保持する属性によって決まる。

● リソースの自動リロードに関する要件

ファイルマネージャは、ゲーム途中でのアーカイブの追加マウントに対応する。

- ➢ 自動リロード用差分オーサリングファイルは、一時的に最高優先度でマウントして処理する。
  - 自動リロードファイル内のファイルは、同名ファイルのキャッシュをすべて破棄する。
  - オブザーバーを通してリソースマネージャに追加されたファイルのリストが渡されると、リソースマネージャはファイルリストに該当するリソースを（リソースがあったら）ロード中状態に戻し、ファイルマネージャに読み込みを再要求する。（リソースの中身だけ削除して読み込み直す）
  - 自動リロードファイルがマウントされている間は、読み込み要求に対して最優先で使用される。
  - マウントしっぱなしだとファイルを上書きできないので、リロードが完了したら自動的にアンマウントする。
  - マウント後、一定時間を過ぎても読み込みの要求がなければアンマウントする。

● ファイルシステムの設定に関する要件

「マウントするファイル」「自動リロードするファイル」「マウントするファイルの優先度」は、あらかじめファイルマネージャの「設定ファイル」に設定しておき、ファイルシステムの起動時に最初に読み込む。

- ➢ 設定ファイルには、例外的にホスト（PC）から読み込むファイルも定義可能。
  - ゲーム上とホスト上のファイルパスの対応付けを列挙する。
- ➢ ホスト上のファイルに関連づけられたファイルは、ゲーム側から書き込みすることもできるため、ゲーム上でエディットしたファイルの書き出しなどの用途に使用するこ

Draft

とができる。

● ファイルマネージャが扱うバッファに関する要件

ファイルマネージャは、ファイルリストバッファと、ファイルの読み込みバッファの二つのバッファを必要とする。

- ファイルリストバッファには、マウントした複数のアーカイブファイル全てのファイルリストを読み込む。
- 読み込みバッファは、読み込み要求に応じてファイルを読み込むバッファである。圧縮ファイルの読み込みバッファとその展開バッファの両方を兼ねる。

● ファイルアクセスの制約に関する要件

ファイルリストのサイズを極力小さく抑え、処理を効率化するために、ファイルアクセスに対する厳しい制約がある。

- ファイルの読み込み要求は必ずフルパス。
  - フルパスの CRC 値によってのみファイルを識別する。
- カレントフォルダの概念がなく、相対パスも使えない。
- フォルダ名、ファイル名の文字列がない。（デバッグ用データとして扱う事は可能）
- フォルダ、ファイルの列挙ができない。

● ファイル読み込み要求に関する要件

ファイルマネージャに対する読み込み要求はキューイングして扱う。

- 要求されたファイルが存在しなければ即時エラーを返す。
- ファイルが存在する場合、ハンドルを返して読み込みを開始する。
- すでに読み込みバッファにファイルが存在する場合、既存のハンドルを返す。
  - オープン中のハンドルの場合、複数の処理が共有してオープンしている状態になる。
  - リソースマネージャの場合は、ファイル読み込み要求を出す前に構築済みのリソースと同じファイルかチェックするので、多重要求をするようなことはない。
  - クローズ済みのファイルバッファが（キャッシュとして）残っている場合、オープン中の状態に戻してハンドルを返す。
    - クローズされたファイルは削除待ちのキューに回され、削除前であれば再利用される。
    - 削除待ちキューは、読み込み要求の発生に伴って削除されていく。
    - ただし、圧縮ファイルはできる限り残す。通常ファイルのバッファだけ削除して新しいファイルのバッファを確保できたなら、削除待ちキューは圧縮ファイルのバッファを掴んだ

Draft

まま削除されずに残る。

- クローズ済みファイルの圧縮ファイルだけが残っている場合、オープン中の状態に戻してハンドルを返し、圧縮ファイルの展開から処理する。

➢ 読み込み要求を出した側の処理は、毎フレーム読み込みの完了状態をチェックし、完了していたらバッファのポインタを受け取って処理を行い、不要になったらクローズする。

➢ 読み込み要求を出す際は、読み込みバッファへの展開ではなく、任意のバッファを指定することが可能。

- 例えば、テクスチャのようなデータはファイルイメージそのまま使用するので、読み込みバッファに読み込むよりも、最初からリソース管理領域に展開した方が効率的。
- 事前に確保すべきメモリのサイズはすぐに調べることができる。（ヘッダーのファイルリストには本来のファイルサイズが記録されているため）
- 任意のバッファを指定する場合は、「メモリ確保→バッファを指定して読み込み要求」の流れでも、「読み込み要求→メモリ確保→バッファ指定」の流れでも良い。
  - ・ 後者の場合、読み込みが始まるまでにメモリ確保を完了してバッファを指定できれば、直接指定バッファに読み込まれる。
  - ・ メモリ確保が遅れて、読み込みが始まったあとにバッファが指定された場合は、読み込み完了後に自動的に指定バッファへの転送が行われる。（転送まで完了して読み込み完了扱いとなる）※極力「読み込みは待たせない」のが基本方針。
- 圧縮ファイルの場合、圧縮ファイルが読み込みバッファに読み込まれた後、指定バッファに直接展開される。
- 読み込みバッファに展開されないファイルはキャッシュとして機能しない。
- 圧縮ファイルは常に読み込みバッファに読み込まれるので、キャッシュとして機能する。

➢ アーカイブ内アーカイブの読み込み要求では、そのアーカイブ内のファイルの数だけいっぺんに読み込みが発生したものとして処理する。

- 例えば、5 つのファイルがあるなら 5 つの読み込み要求を同時に発生させ、1 回の読み込みで全ての読み込みを完了させる。
- 読み込みが完了しても、実際に読み込みがなければそのまま削除待ちのキューに回される。
- 対象ファイルの中に読み込み済み（キャッシュされている）ファイルがある場合は読み込まれない。
- 対象ファイルの中に、「優先度の高いアーカイブ」に含まれるファイルがあれば、そちらから読み込まれる。
- このアーカイブ内アーカイブの仕組みにより、同じファイルが複数の場所に配置される可能性がある。
  - ・ これにより、読み込みを効率化する。
  - ・ 分散するファイルが全て同じ内容であることは、アーカイブ時にアーカイブツールによっ

Draft

て確認・保証される。

・ どこに配置されたファイルであっても、同じパス（の CRC）のファイルであればキャッシュとして有効。

● マルチスレッド処理に関する要件

ファイルマネージャは専用スレッドで稼働し、バックグラントでのファイル読み込みと圧縮ファイルの展開を行う。

- スレッドの優先度はゲームよりも低く設定する。
- 現在のシーク位置に基づいて読み込みをスケジューリングし、位置的に近いファイルを優先的に読み込む。
- 圧縮展開処理は、別途「ジョブ」を投入して並行処理で実行する。
  - 「ジョブ」については、別紙の「「サービス」によるマルチスレッドの効率化」を参照。
- 読み込み要求がない時は、スリープして待機し、キューイングされるのを待つ。
  - ストリーミングを扱う場合、スリープできるようなタイミングはほとんどない。

● 【できれば】ストリーミングに関する要件

ストリーミングを扱う場合、外部から渡されたリングバッファに対する読み込みに対応する。

- リングバッファの残量を逐一チェックし、しきい値未満になったら優先的に読み込むようにスケジューリングする。
- ストリーミングの読み込みを行う際は、一時的にゲームよりもスレッド優先度を高くする。
- ストリーミングの読み込みには三つのタイプがある。
  - BGM のようなループタイプ（BGM でも 1 ループで完了するものもある）
  - ムービーやボイスのような 1 ループで完了タイプ（ループするムービーもある）
  - 即時再生のための予約バッファリング（予約バッファリングにはファイル読み込みバッファが使用され、再生が確定した際にメモリを転送する）
- 全てのファイルアクセスを一つのファイルマネージャで処理しないと、効率的なシークのスケジューリングができないので、できる限り、ストリーミングも含めて対応する。

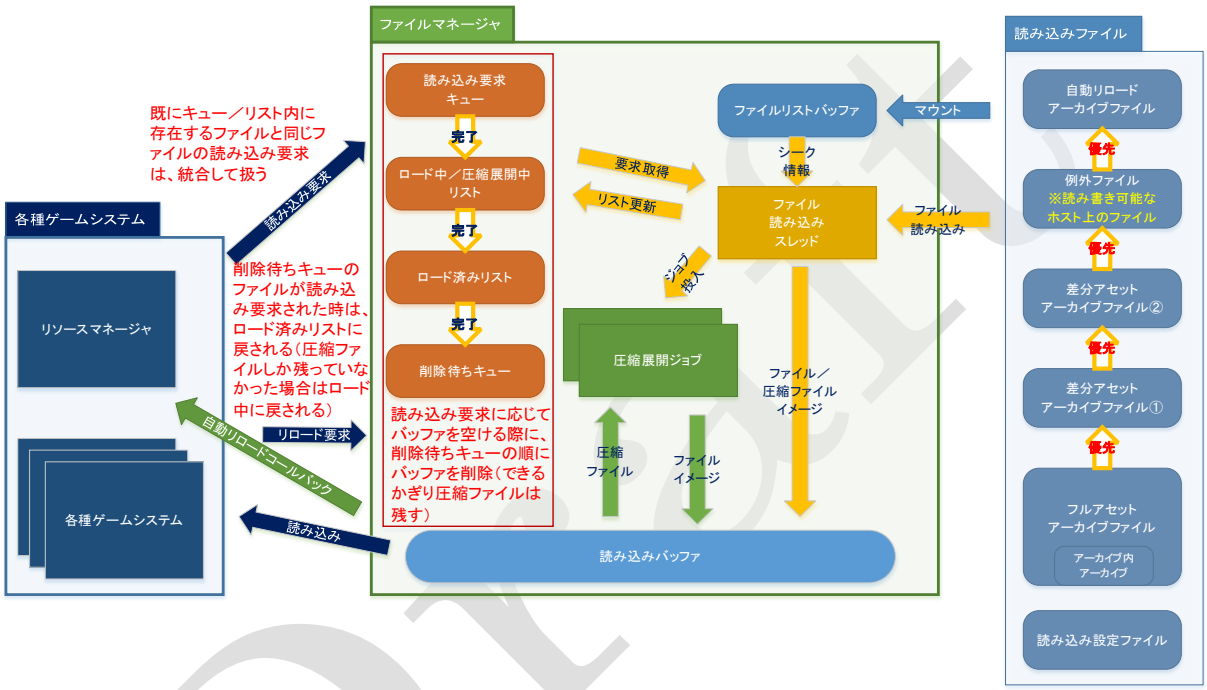
Draft

■ 仕様概要

▼ システム構成図

要件に基づくシステム構成図を示す。

ファイルマネージャのシステム構成図：



▼ アーカイブツール

専用のアーカイブツールを使用してファイルをアーカイブする。

アーカイブツールは下記の処理に対応する。

- 指定のフォルダ以下のファイルをまとめてアーカイブする。
- 基本フォルダをルートフォルダに見立てたフルパスでファイルを管理する。
- ファイルパスは全て小文字化した上で、CRC 化して管理する。
- 基本的に、ファイルごとに圧縮したファイルを扱う。
  - 非圧縮の指定も可能。
  - 圧縮効率のファイルは自動的に非圧縮扱いにする。
- ヘッダー部にファイルリストを列挙する。
  - ファイルリストは、1 ファイルごとに「ファイルパスの CRC」「ファイル拡張子の CRC」「ファイル位置」「圧縮ファイルサイズ」「本来のファイルサイズ」を扱う。4 バイト×5

で、1 件当たり 20 バイト。
  - ファイルパスの CRC の昇順で配置し、そのままメモリに展開してバイナリサーチ可能な構造にする。
- ファイル内にアーカイブファイルを含むことを可能とする。
- ファイル内アーカイブ内のファイルも、ヘッダー部のファイルリストに共に列挙する。
  - 「ファイル位置」はアーカイブ内アーカイブの先頭を指す。
    - アーカイブ内アーカイブ内のファイル読み込み要求時は、アーカイブ内アーカイブ内のファイルをまとめて読み込むため。
  - 「圧縮ファイルサイズ」「本来のファイルサイズ」は対象ファイルのものを指す。
    - どのようなファイルであっても、ファイルを読み込まずにファイルサイズを返すことができるようにするため。
  - これにより、同じパス（CRC）のファイルが列挙される可能性があるが、「ファイル位置」順に全て列挙する。
    - 実際の読み込み時に、シーク位置に応じてもっとも近いファイルが選ばれる。
  - アーカイブ内アーカイブファイルの属性が、「親アーカイブへの展開を許可しない」ものになっている場合は展開しない。
- ファイル内アーカイブファイルは圧縮しない。その中のファイルは圧縮可。
  - 要は、複数のファイルをひとまとめに圧縮することには対応しない。
  - 実機上では常にファイルを単独で管理し、個別の再利用性を高める。
- フッター部に CRC に対するファイルパスの文字列情報を保存する。
- 異なるファイルで CRC が競合するものがあったらエラー。
  - ファイル名を変更することで対処することになる。
  - ゲーム内で使用する全てのアーカイブファイルで CRC の競合があっては行けないので、複数のアーカイブファイルで CRC の競合がないかチェックする機能もツールに設ける。
    - フッター部のファイルパス情報を利用して、競合を判定する。（「CRC の一致＝ファイルパスの一致」なら問題なし）
- ファイル内アーカイブの仕組みにより、同じファイルで内容が異なるものがあったらエラー。
  - ツールの処理過程で、内部的にファイル内容を MD5 などのハッシュ化した情報を記録して照合する。
- 非圧縮状態で 2GB を超えるファイルがあったらエラー。
- アーカイブファイル全体（ヘッダー＋ファイル内容＋フッター）で 4GB を超えたらエラー。
- アーカイブ時に、ヘッダー情報のためのエンディアンを指定可能。
- アーカイブ時に、「親アーカイブへの展開を許可しない」属性を指定可能。

Draft

- この場合でもアーカイブ内アーカイブとして処理する際には一つ一つのファイルをチェックし、CRC の競合、ファイル内容の不一致をチェックする。

・ アーカイブ時には、任意のフォルダに配置されている設定ファイルを読み込んで処理する。

- 設定ファイルは所定のファイル名で随所に配置される。
- 設定ファイルにより「アーカイブ内アーカイブとしてまとめるファイルの指定」や「ファイル毎の圧縮の有無の指定」などが記述され、設定ファイル以下のサブフォルダに適用される。

▼ アーカイブファイル内アーカイブについて

要件定義で示した「アーカイブ内アーカイブ内のファイルは、親のアーカイブのファイルリストにも登録される」について補足する。

具体的には下記のようなイメージである。

アーカイブファイル:「/data/chara/x0010.arc」のイメージ

位置	ヘッダー部　※実際はファイルはCRC順に並ぶ			
	ファイルパス　※実際はCRC	ファイル位置	圧縮ファイルサイズ	本来のファイルサイズ
16	/data/chara/x0010/x0010.mdl	128	12,046	21,682
32	/data/chara/x0010/x0010.mot	12,174	48,238	86,828
48	/data/chara/x0010/x0010.tex	60,412	56,864	102,355
64	/data/chara/x0010/x0010.cfg	117,276	0	268
80	/data/chara/common/common.tex	117,276	25,368	45,662
96	/data/sound/x0010/x0010.se	142,644	32,588	58,658
	データ部			
128	/data/chara/x0010/x0010.mdl のファイル内容			
12,174	/data/chara/x0010/x0010.mot のファイル内容			
60,412	/data/chara/x0010/x0010.tex のファイル内容			
117,276	/data/chara/x0010/x0010.cfg のファイル内容			
117,276	/data/chara/common/common.tex のファイル内容			
142,644	/data/sound/x0010/x0010.se のファイル内容			

Draft

↓　※このファイルを別のアーカイブファイルに含めたイメージ

アーカイブファイル：「/full.arc」のイメージ

位置	ヘッダー部　※実際はファイルはCRC順に並ぶ ファイルパス　※実際はCRC	ファイル位置	圧縮ファイルサイズ	本来のファイルサイズ
16	/data/other/xxx.x	128	56,236	101,224
	…（略）…			
1,024	/data/chara/chara/x0010.arc	65,536	0	175,232
1,040	/data/chara/x0010/x0010.mdl	65,536	12,046	21,682
1,056	/data/chara/x0010/x0010.mot	65,536	48,238	86,828
1,072	/data/chara/x0010/x0010.tex	65,536	56,864	102,355
1,088	/data/chara/x0010/x0010.cfg	65,536	0	268
1,104	/data/chara/common/common.tex	65,536	25,368	45,662
1,120	/data/sound/x0010/x0010.se	65,536	32,588	58,658
	…（略）…			
65,520	/data/zzz/zzz.x	1,025,684	12,046	21,682
	データ部			
128	/data/chara/x0010/x0010.mdl のファイル内容			
	…（略）…			
65,536	/data/chara/x0010.arc のファイル内容			
	…（略）…			
1,025,684	/data/zzz/zzz.x のファイル内容			

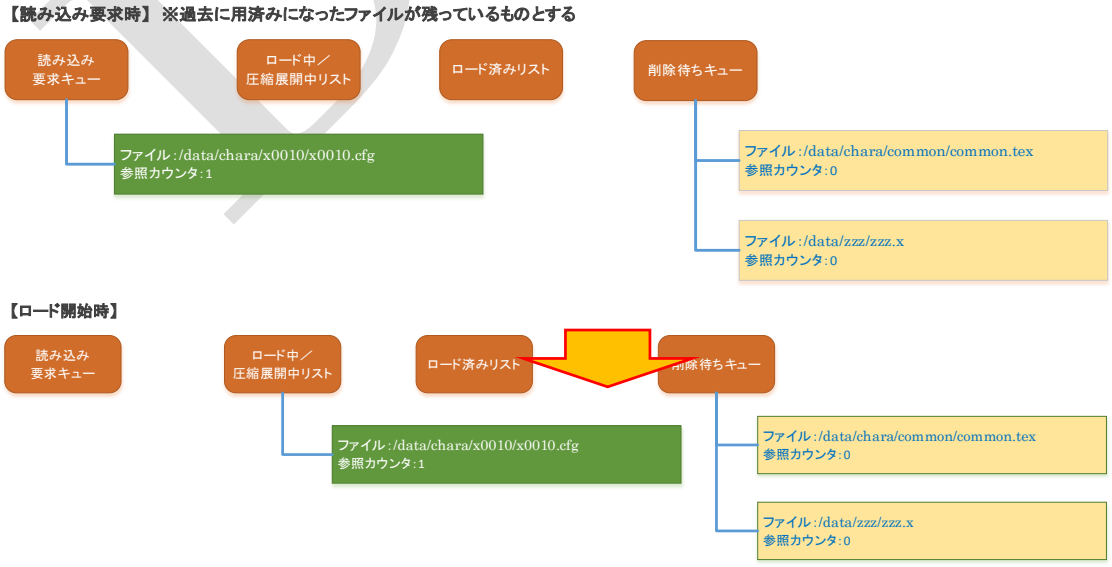
※/data/chara/x0010/x0010.arc のファイル内容がヘッダー部に展開される。

この「full.arc」をマウントした状態で、例えば「/data/chara/x0010/x0010.cfg」の読み込み要求を行うと、本来アーカイブファイル内のファイルであるが、ヘッダー部の情報に基づいて、読み込むべきファイルを探り当てることができる。

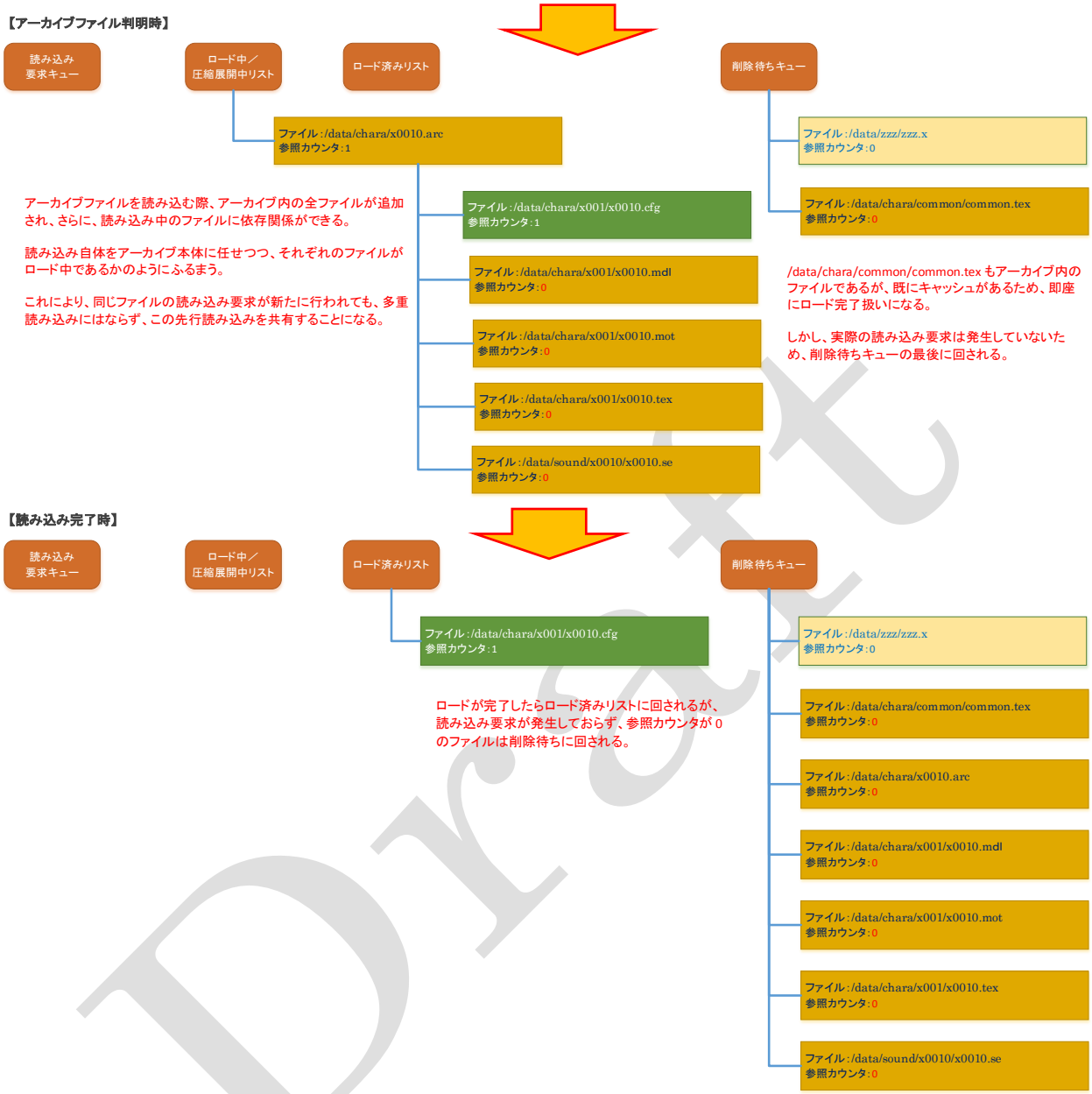
さらに、「ファイル位置」を参照した先のファイルがアーカイブファイルであるため、そのアーカイブファイルのファイルをまとめて読み込む。

それは、アーカイブファイル内のファイル読み込みが指定されたということが、「そのアーカイブ内のファイル全てが必要なタイミング」とみなす（予測する）のが、このファイルシステムの考え方である。

この時、処理内部の読み込みリクエストは下記のように動作する。



Draft



▼ ファイル読み込みバッファについて

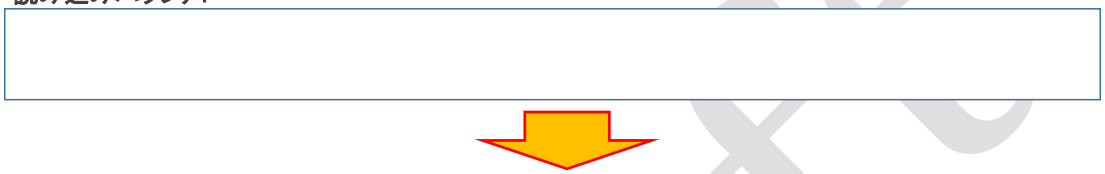
ファイル読み込みバッファは、圧縮ファイル、および圧縮展開後のファイルイメージを配置する大きなバッファである。

Draft

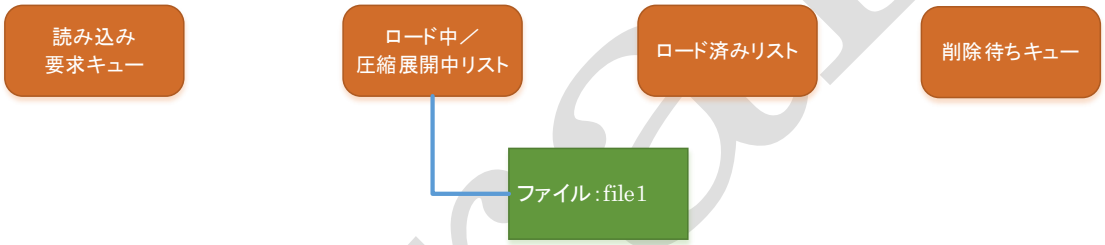
読み込みバッファの使用イメージを示す。

【読み込み要求時】

読み込みバッファ：



【ロード中】

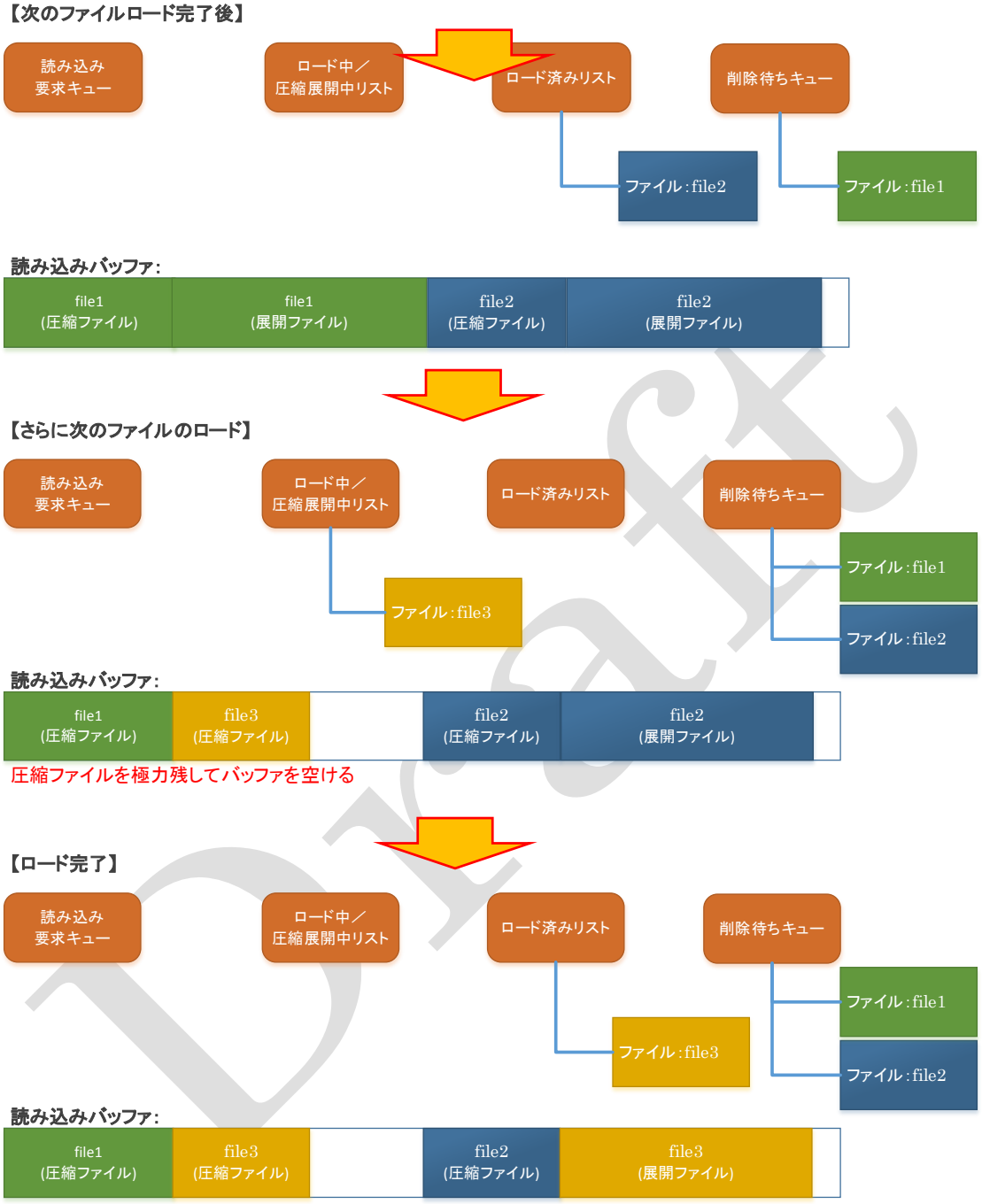


読み込みバッファ：

※実際の読み込みバッファのサイズはもっと巨大

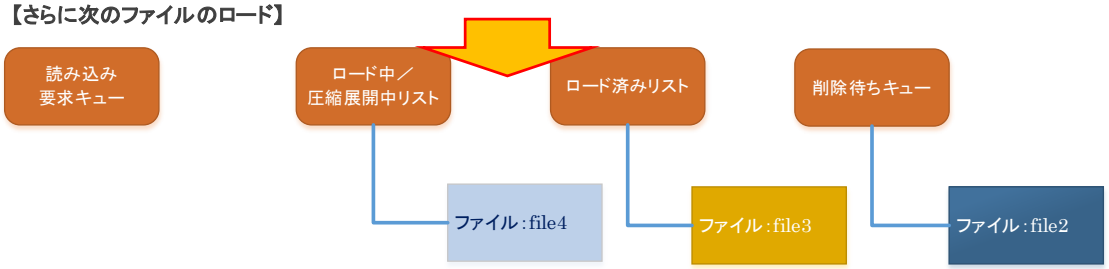
【ロード完了後】

読み込みバッファ：


【次のファイルロード完了後】
読み込み
要求キュー
ロード中／
圧縮展開中リスト
ロード済みリスト
削除待ちキュー
ファイル：file2
ファイル：file1
読み込みバッファ：
file1
(圧縮ファイル)
file1
(展開ファイル)
file2
(圧縮ファイル)
file2
(展開ファイル)
【さらに次のファイルのロード】
読み込み
要求キュー
ロード中／
圧縮展開中リスト
ロード済みリスト
削除待ちキュー
ファイル：file3
ファイル：file1
ファイル：file2
読み込みバッファ：
file1
(圧縮ファイル)
file3
(圧縮ファイル)
file2
(圧縮ファイル)
file2
(展開ファイル)
圧縮ファイルを極力残してバッファを空ける
【ロード完了】
読み込み
要求キュー
ロード中／
圧縮展開中リスト
ロード済みリスト
削除待ちキュー
ファイル：file3
ファイル：file1
ファイル：file2
読み込みバッファ：
file1
(圧縮ファイル)
file3
(圧縮ファイル)
file2
(圧縮ファイル)
file3
(展開ファイル)

Draft

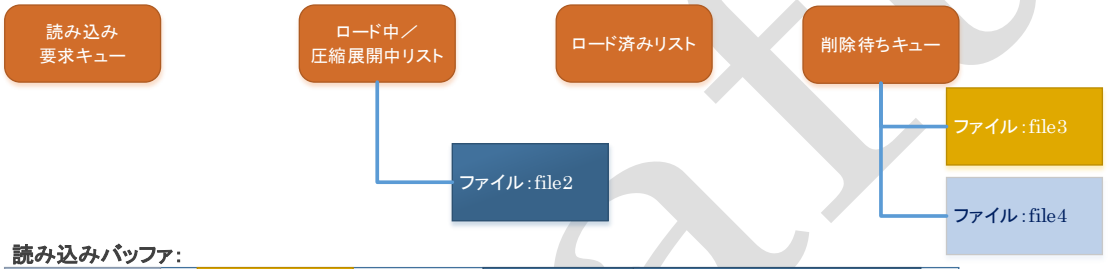
【さらに次のファイルのロード】



読み込みバッファ：

file1が削除

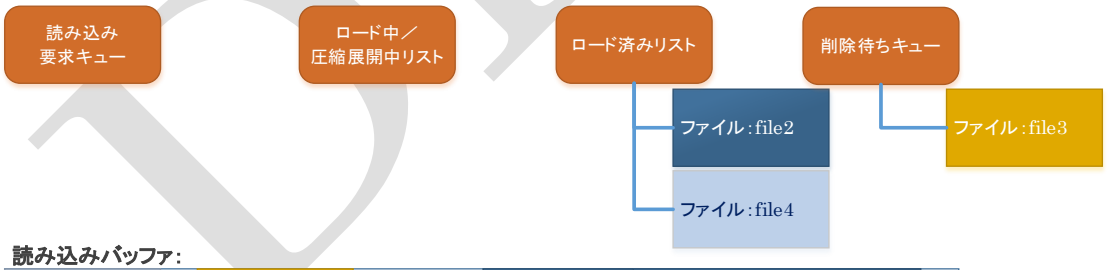
【file2の再読込要求】



読み込みバッファ：

file2は圧縮ファイルが残っていたので、キューイングするまでもなくロード中に移し、圧縮ファイルの展開のみ実行

【file4の再読込要求】



読み込みバッファ：

file4はファイルイメージが残っているので、即ロード済みに

Draft

■ データ仕様

▼ アーカイブファイル

アーカイブファイルのデータ構造を示す。

項目	型	サイズ	内容
01：ヘッダー部			
マジックナンバー	i8[4]	4	'f', 'S', 0x00, 0xfd
エンディアン	u8	1	0 … リトルエンディアン、1 … ビッグエンディアン
親アーカイブに展開	u8	1	0 … 禁止、1 … 許可
（4バイトパディング）	u8[2]	2	0xfd で埋める
総サイズ	u32	4	ヘッダー部も含む全体サイズ（バイト） フッター部は含まない
総サイズ（デバッグデータ含む）	u32	4	ヘッダー部とフッター部も含む全体サイズ（バイト）
データ部オフセット	u32	4	ヘッダー部の先頭からのオフセット値（バイト）
フッター部オフセット	u32	4	（同上）
（32バイトパディング）	u8[8]	8	0xfd で埋める
		(32)	
02：ファイルリスト部（通称ヘッダー部）			
ファイル数	u32	4	
ファイル情報[n]			
ファイルパスCRC	u32	4	全ての文字を小文字化し、セパレータを '/' にしたファイルパスのCRC
ファイル拡張子CRC	u32	4	全ての文字を小文字化し、最も長い拡張子が扱われる。例えば、file.ext.more なら .ext.more が拡張子。
ファイル位置	u32	4	ファイルの先頭からの絶対位置（そのままシーク位置として扱う）
圧縮ファイルサイズ	u32	4	非圧縮ファイルなら 0
本来のファイルサイズ	u32	4	
※ファイル数分続く…			
		(4 + 20 * n)	
03：ファイル部			
ファイル内容[n]			パディングなしで列挙
ファイル内容	u8[]	?	
※ファイル数分続く…			
（16バイトアラインメントパディング）	u8[]	?	0xfdで埋める
		(?)	
04：終端部			
マジックナンバー	i8[4]	4	0xfb, 'F', 's', 0x00
（16バイトアラインメントパディング）	u8[12]	12	0xfd で埋める
		(16)	
05：ファイルパス：参照情報部（通称「フッター部」）			
ファイルパス位置情報[n]			
ファイルパス文字列情報の位置	u32	4	ファイルパス：参照情報部の先頭からのオフセット位置
※ファイル数分続く…			
		(4 * n)	

※「05：ファイルパス：参照情報部」～「06：ファイルパス：文字列情報部」を切り出してメモリにコピーして使用するため、
位置情報はこの先頭からのオフセット位置を扱う。

06：ファイルパス：文字列情報部（通称「フッター部」）			
ファイルパス文字列情報[n]			
親フォルダ情報の位置	u32	4	ファイルパス：参照情報部の先頭からのオフセット位置 ※0でルートフォルダ
ファイル名／フォルダ名文字列	u8[]	?	文字列データ
（4バイトアラインメントパディング）	u8[]	?	0xfdで埋める
※フォルダとファイル数分続く…			
		(? * n)	

※ファイル名、フォルダ名、サブフォルダ名をバラバラに扱う。
例えば、「/dir/sub/file1.txt」と「/dir/file2.txt」というファイルなら、[0:dir] [16:sub] [20:file1.txt] [16:file2.txt] のように分かれる。

07：最終端部			
（16バイトアラインメントパディング）	u8[?]	?	0xfd で埋める
		(?)	

▼ ファイルシステムの設定ファイル

「設定ファイル」は、ゲーム起動時に最初に読み込まれるファイルである。

ROM 上のルートフォルダ、もしくは、ホスト上のホームフォルダに配置する所定のパスのテキストファイル（JSON 形式）であり、ゲームで使用するアーカイブファイルを列挙する。このファイルをバイナリ変換したファイルを読み込む。

設定ファイルの具体的なイメージは下記の通り。

設定ファイルのサンプル：

【fsys.json】

```
//ファイルシステム設定ファイル
{
    //マウントするアーカイブファイル
    "mound":
    [
        //優先度が高い順にアーカイブファイルを列挙する
        { "arc": "/user_x.arc" },//個人制作分の差分アーカイブ
        { "arc": "/team_a.arc" },//別チームから一時的に受け取った差分アーカイブ
        { "arc": "/full02.arc" },//フルアセットのアーカイブ 02
        { "arc": "/full01.arc" }//フルアセットのアーカイブ 01
    ],

    //自動リロード用アーカイブファイル
    "autoReload": "/user_a_new.arc",

    //ホストファイルの関連付け
    //※ゲーム上でデータをエディットして出力するような場合に利用
    //※実機上ではフォルダ／ファイルを列挙することができないため、
    //  ワイルドカードが使えない
    "map":
    [
        { "path": "/data/stage/01/light001.cfg",  "host": "/out/light001.cfg" },
        { "path": "/data/stage/02/light002.cfg",  "host": "/out/light002.cfg" },
        { "path": "/data/stage/03/geom03.txt",    "host": "/out/geom03.txt" },
    ]
}
```

▼ アーカイブ設定ファイル

アーカイブファイル作成時は、指定のフォルダ以下のファイルをまるごとアーカイブするのが基本である。

この時、アーカイブを作成しながら子アーカイブを作成したり、一部のファイルを圧縮しないように指定したりといったことができる。「アーカイブ設定ファイル」を用意し、随所のフォルダに配置することで、そのような設定を行う。

「アーカイブ設定ファイル」は、特定の名前のテキストファイル（JSON 形式）で扱う。

アーカイブ設定ファイルのサンプル：

【/data/char/x0010/arc.json】

```
//アーカイブ設定ファイル
{
    //子アーカイブファイルを作成
```

Draft

```
    "childArc":
    [
        {
            "arc": "../x0010.arc",
            "files":
            [
                "x0010.mdl",
                "x0010.mot",
                "x0010.tex",
                "x0010.cfg",
                "/data/char/common/common.tex"
            ]
        }
    ],
    //アーカイブファイル属性
    "arcAttr":
    [
        { "arc": "../x0010.arc", "attr": "winthin" }//親フォルダのファイルセットに組み込む　※デフォルト
                                                    //組み込まない場合は "without" を指定
    ],
    //非圧縮ファイル設定　※デフォルトは圧縮する
    "nocomp":
    [
        "x0010.cfg"
    ]
}
```

【/data/sound/x0010/arc.json】

```
//アーカイブ設定ファイル
{
    //他チームの子アーカイブファイルに組み込む
    "childArc":
    {
        "arc": "/data/chara/x0010.arc",
        "files":
        [
            "x0010.se"
        ]
    }
}
```

以上のように、多数のチームが作ったファイルを一つの子アーカイブに組み込みたい場合、それぞれのチームで担当分のファイルの扱いを設定する。このような構造をとることにより、フォルダごとの責任の所在を明確にする。（一つのフォルダを複数のチームで共通利用しない）

一つのファイルを多数の子アーカイブに組み込むように設定することも可能。

なお、子アーカイブに組み込まれたファイルは、通常ファイルとしてはアーカイブされなくなる。

Draft

■ 処理仕様

▼ アーカイブツール

・ 開発言語：C#コンソールアプリケーション（.Net Framework4.0 以上）
　　選定理由： 開発のし易さ、JSON パーサーの利用、dynamic 型（遅延バインド）の利用。
・ 使用ライブラリ：JSON. Net
　　http://json.codeplex.com/downloads/get/744406
・ ツール名：narc.exe　※Nested ARChiver の意。
・ 使用方法：（例）

```
$ narc.exe --be --within --comp 60 --base c:¥work¥data -I
  c:¥work¥data¥chara¥ -o c:¥work¥user_a.arc
```

【対応オプション】　※競合オプションは後に指定されたものが有効

--le ..... リトルエンディアン　※デフォルト
--be ..... ビッグエンディアン
--within ... 親アーカイブへのファイルリスト展開を許可　※デフォルト
--without .. 親アーカイブへのファイルリスト展開を禁止
--comp % ... 圧縮ファイルを許可する圧縮率（指定 % 以下の圧縮率の時しか圧縮を許可しない）　※デフォルト = 50%
--base dir . 基本フォルダ指定
-i dir ..... アーカイブファイル作成対象フォルダ指定
-o file .... アーカイブファイル（出力ファイル）指定

▼ ファイルマネージャ

【構想】

ファイルマネージャの処理は、ファイルアクセスのインターフェースとなる「ファイルマネージャ」に加えて、バックグラウンドでの「ファイル読み込みスレッド」「圧縮ファイル展開ジョブ」で構成される。

■■以上■■

Draft

■ 索引

索引項目が見つかりません。

Draft

開発を効率化するためファイルシステム

以　上

Draft