ゲームデータ仕様

– ゲームデータのフォーマットと変換 –

2014 年 1 月 10 日 初版

板垣 衛

■ 改訂履歴

版	リリース	担当	改訂内容
初版	2014 年 1 月 10 日	板垣 衛	（初版）

■ 目次

■ 概略・目的

本書における「ゲームデータ」とは、大まかには、グラフィック関係データとサウンド関係データ以外のデータ全般を指す。

その多くはプランナーが扱うデータで、ゲームを制御するための設定やパラメータなどのことである。多彩なデータを扱い、ゲーム固有のデータ構造となるものが多い。

本書は、「ゲームデータ」の入力フォーマットと実機上のデータフォーマット、および、その変換・取り込み処理に関する基本仕様を規定する。

■ 基本用語

▼「ゲームデータ」

本書においては、「ゲームデータ」という用語を下記の意味で扱う。

- グラフィック関係データとサウンド関係データを除くデータ全般。一般的には、これらのデータを含むリソース全般をまとめて「ゲームデータ」と呼ぶが、本書では区別して扱う。
- グラフィック関係データとサウンド関係データであっても、それを制御するためのゲーム固有のデータは「ゲームデータ」に類する。
- 何らかの処理設定やデバッグ用データなど、プログラマーが扱うデータもまた「ゲームデータ」である。
- 基本的には、予め定義された静的なデータを指し、ファイル（リソース）として扱われる。
- メモリ上で内容が変動する動的なデータやセーブデータなどは、「ゲームデータ」の範疇に含まない。しかし、例えば「ゲームデータを読み込んでセーブデータを復元する」といった、動的なデータを再現するためのゲームデータの活用はありえる。
- PlayStation 系では「インストールデータ」を指して「ゲームデータ」と呼ぶが、本書における「ゲームデータ」はそれとは別物である。

■ 要件定義

▼ 基本要件

- ゲームデータの入力データをテキストファイル形式で扱う。
- Excel や DB で管理するデータは、テキストファイルに変換して扱う。なお、この要件については本書の範疇外とし、専用の仕様として別途策定する。
- テキストファイルをバイナリデータに変換する汎用ツール（パーサー）を作成する。
- バイナリデータを実機に取り込む処理を作成する。
- テキストのファイルのパーサーは、実機上では扱わない。

▼ 要件定義

- テキストファイルは、JSON を基本フォーマットとして統一する。
- テキストファイルの文字コードは UTF-8 とし、日本語や欧州文字に対応する。
- テキストファイルは、下記の拡張仕様（JSON が非対応の仕様）に対応する。
  - JavaScript 形式のコメント文を使用できる。（例：// comment、/* comment */）
  - C 言語形式の#include 文と#define 文を使用できる。
  - データ部に四則演算を用いることができる。（例：{ ”age”: 30 + 3, … } ）
  - データ部に CRC 変換やゲーム用計算式変換などの特殊な関数を使用できる。（例：{ ”id”: CRC(“c0010”), … }, { “condition”: Expr(“IsFlag(¥“AlreadyMetOldMan¥”) == True && GetChapter() >= 2”), … } ）

  注：JSON データを MongoDB などのドキュメント指向データベース（BSON 形式で保存される）で扱う場合、これらの拡張仕様が使えないので注意。同様の情報を扱うための別の仕様も合わせて策定する。
- テキストファイルからバイナリデータに変換するための変換設定もまた JSON 形式のテキストデータとして定義する。期待される値の範囲など、エラー判定用の設定も可能。
- 不定長の配列をメンバーに持つ構造体にも対応。
- 専用のデータ変換ツールを通して、バイナリデータを出力する。
- データ変換ツールは、下記の仕様に対応する。
  - CUI ツールとして構成し、単純に一つのテキストファイルを一つのバイナリファイルに変換する。これは、任意のバッチ処理や他のツールからの呼び出しなどに対応しやすい形式である。
  - エンディアンの指定、ポインターのビット数（32 or 64）指定に対応。

- データ構造／内容のエラーを検出した場合、出力ファイルは作成されず、エラーが通知される。

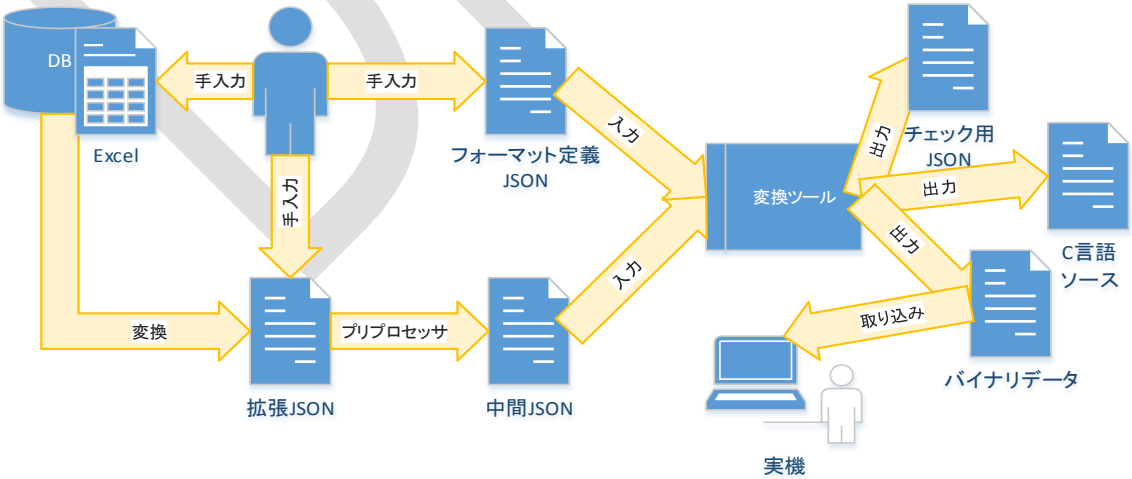
・ バイナリデータは、メモリ上のイメージとしてほぼそのまま取り込める。文字列のポインター変換などの処理も取り込み時に同時に行われる。
・ バイナリデータの構造が変更され、実機上の構造とずれた場合、取り込み処理は自動的にその事を検出し、項目毎のデータ取り込み処理を行う。要は、データ構造が変わっても、プログラムが極力正常に動作するようにする。

■ 仕様概略

▼ 環境

・ OS：Windows 系 PC（XP 以上）32bit/64bit
・ 必須ツール①：テキストエディタ　※なんでもよい
・ 必須ツール②：MinGW(GCC)　※プリプロセッサ
・ 必須ツール③：変換ツール　※独自開発
・ 使用ツール（オプション）：Python + SCons

▼ ワークフロー



・ DB／Excel ..... 元データ　※この仕様書では扱わない
・ 拡張 JSON ..... テキストデータ（JSON 形式＋JavaScript 形式コメント

＋C 言語形式#include 文・#define 文＋四則演算式＋特殊関数）

- 中間 JSON ..... コメント除去、#include 文・#define 文展開
- フォーマット定義 JSON ... データ変換・ルール・バージョン定義
- 変換ツール ..... JSON データをバイナリデータに変換（四則演算、特殊関数計算も行う）
- バイナリデータ ..... バイナリデータ（C 言語構造体と一致）
- C 言語ソース ..... 構造体とデータフォーマット定義（オプションで出力、フォーマット定義 JSON のみに基づいて生成）
- チェック用 JSON ..... 変換済みデータと同じ内容の JSON（内容確認用）
- 実機 ..... ゲーム実行時にバイナリデータ取り込み

■ データ仕様

▼ DB/Excel

この仕様書では扱わない。別途仕様を策定。

想定としては、ドキュメント指向 DB（MongoDB など）＋RDB（PostgreSQL など）で管理。Excel のインポート／エクスポートでデータ編集。独自のバージョン管理とロック機構を備えた仕組みを構築。ツールのインターフェースは Web 形式。

なお、DB に記録される JSON は、一切の拡張仕様が使用できないが、直接テキストを編集しないのでほぼ問題ない。特殊関数の使用が必要な箇所は、フォーマット定義 JSON 側に関数の使用を定義する。

DB 内に記録されるデータ構造は、Excel 形式への変換に特化しているため、実機向けのデータ構造と異なる。一つの DB データから、複数の実機向けデータを出力する場合もある。

▼ 拡張 JSON

基本 JSON 仕様：

▼ 中間 JSON

フォーマットは拡張 JSON とほぼ同じ。コメントが除去され、#include 文と#define 文

が展開されているだけの違い。エラー出力用に、#line 文が挿入されている。MinGW(GCC)による C++言語プリプロセッサの使用を想定。

▼ フォーマット定義 JSON

▼ バイナリデータ

▼ C 言語ソース　※オプションで生成

▼ チェック用 JSON

バイナリ出力が成功した時にだけ出力される。バイナリデータの構造に合わせた構造。完全な JSON 仕様を踏襲したフォーマットのため、データの二次利用も可能。

■ 処理仕様

▼ プリプロセッサ

MinGW(GCC)

http://sourceforge.net/projects/mingw/files/Installer/

▼ データ変換ツール

▼ 実機（取り込み処理）

■ 環境の改善

▼ SCons の利用

SCons

Python

以上

■ 索引

ゲームデータ仕様

以　上