プログラム設計仕様書：バイナリデータ変換ツール

2025年05月18日
作成／改定 ： 土屋 明生

1. 概要

本プログラムは、メインフレーム等で利用されるEBCDICおよびJEFエンコードされた固定長バイナリデータファイルを、そのデータ構造を定義したCOPY句ファイルに基づいて解析し、人間が可読なUTF-8エンコードされたCSV形式のDATファイルに変換するツールです。プログラムはクラス構造を採用し、COPY句解析、データ変換、ファイル入出力の各機能をカプセル化しています。

主な機能として、文字エンコーディングの変換（EBCDIC→ASCII/UTF-8互換、JEF→Unicode変換定義利用）、数値データ形式の変換（ゾーン10進数、パック10進数→数値文字列/数値）を行います。

2. 改訂履歴

日付	バージョン	作成者/改訂者	改定内容
2025/03/01	1.0	土屋明生	初版作成
2025/05/18	1.1	土屋明生	クラス構造化、関数名変更、JEF変換定義利用の明確化、COMP-3変換統合、CSVエスケープ処理反映など

3. 機能一覧

- 指定ディレクトリ内の特定の拡張子を持つバイナリファイルを一括処理。
- 各バイナリファイルに対応するCOPY句ファイルを読み込み、データ構造を解釈。
- 外部指定された文字コード変換定義ファイル（JEF→Unicodeマップ）を読み込み、変換処理に利用。
- COPY句の定義に基づき、バイナリデータから各フィールドを抽出。
- フィールドごとにデータ型に応じた変換処理を実行：
    - EBCDIC文字 (PIC X) からASCII互換Unicode文字列への変換（NULL除去、前後トリム）。
    - EBCDIC数値 (PIC 9, ゾーン10進数) から数値文字列への変換（NULL除去、前後トリム、先行ゼロ除去）。
    - 日本語文字列 (PIC N, JEF想定) からUTF-8文字列への変換（JEF変換定義マップ利用、JEF全角文字置換、未定義文字置換、前後トリム）。
    - ゾーン10進数 (PIC 9...V9...型, V9型) から小数点付き数値文字列への変換（EBCDIC変換、小数点挿入、前後トリム）。
    - パック10進数 (COMP-3, PIC P9/PS9/S9/SP9/PV9/PSV9等) から符号および小数点位置を考慮した数値文字列への変換。
- 変換結果をUTF-8形式のCSVファイルとして指定ディレクトリに出力。
- 出力CSVにおいて、フィールド値内のダブルクォートをエスケープし、フィールド値をダブルクォートで囲む。
- 処理の進捗、結果、エラー情報を標準エラー出力に表示。
- 処理したファイル数、成功/失敗数を集計し、終了時に結果を表示。

4. 入力

4.1. バイナリデータファイル

- 説明: 変換対象のメインフレームデータ。EBCDICや各種数値形式でエンコードされています。
- 格納場所: `./iBIN` ディレクトリ (設定可能)。
- 命名規則: 任意のファイル名。拡張子は `.bin` (設定可能)。
- 例: `TRANSDATA.bin`, `MASTER01.bin`
- フォーマット: 固定長レコード。レコード長は対応するCOPY句ファイルで定義されます。

4.2. COPY句ファイル

- 説明: バイナリデータのレコードレイアウトを定義するテキストファイル。COBOLのCOPY句に類似した形式。
- 格納場所: `./iCPY` ディレクトリ (設定可能)。
- 命名規則: `CPY_{バイナリファイル名(拡張子なし)}.txt` である必要があります。
    - 例: バイナリファイルが `TRANSDATA.bin` の場合、`CPY_TRANSDATA.txt`
- 文字エンコーディング: UTF-8 (BOMなしを推奨)。
- フォーマット:
    - 1行目: レコード全体のバイト長 (数値のみ)。
        - 例: `256`
    - 2行目: ヘッダー行 (未使用、空行または任意の文字列)。プログラムでは無視されます。
        - 例: `(ヘッダー情報など、この行はプログラムでは使用されません)`
    - 3行目以降: 各フィールド定義。カンマ区切りで以下の情報を記載。
        - `項目名,データ型,数値属性,バイト長,オフセット(1ベース)`

列番号	内容	説明	例
1	項目名	DATファイルに出力される際のヘッダー名	`USER_ID`, `ITEM_NAME`
2	データ型	フィールドのデータ型を示す識別子 (下記「6.1. データ型と変換ルール」参照)	`X`, `9`, `N`, `P9(7)V9(2)`
3	数値属性	データ型が `PV9` または `PSV9` の場合の小数点以下の桁数。それ以外は空欄可。数値でない場合は無視。	`2` (PV9, PSV9の場合), (空欄)
4	バイト長	当該フィールドがバイナリデータ中で占めるバイト数	`10`, `8`, `4`
5	オフセット	レコード先頭からの当該フィールドの開始位置 (1から始まる)。	`1`, `11`, `19`

COPY句ファイル記述例 (CPY_SAMPLE.txt):

```
Plaintext
150
(ヘッダー情報など、この行はプログラムでは使用されません)
社員番号,X,,8,1
氏名カナ,N,,20,9
部署コード,9,,4,29
基本給,PS9(7),,4,33
手当,P9(5)V9(2),,4,37
評価ポイント,V9,,3,41
備考,X,,100,44
レコード種別,9,,1,144
更新日,9(8),,8,145
```

- 数値属性が空欄の場合は、カンマ区切りによって列数がずれないよう注意が必要です。例: `項目名,データ型,,バイト長,オフセット`

4.3. 文字コード変換定義ファイル

- 説明: JEFバイト列からUnicodeへのマッピングを定義するファイル。
- 格納場所: プログラム実行時にパスを標準入力で指定します。
- 命名規則: 任意。
- 文字エンコーディング: UTF-16。
- フォーマット: カンマ区切り (CSV)。
    - 各行: `source_hex,target_hex`
        - `source_hex`: 変換元JEF文字の2バイト16進表現 (例: `4040`)
        - `target_hex`: 変換先Unicodeコードポイントの16進表現 (例: `3000`)
    - コメント行は `#` で開始します。

5. 出力

5.1. DATファイル

- 説明: 変換後のデータを格納するCSVファイル。
- 格納場所: `./oDAT` ディレクトリ (設定可能、存在しない場合は自動作成)。
- 命名規則: `LOAD_{バイナリファイル名(拡張子なし)}.dat`
    - 例: バイナリファイルが `TRANSDATA.bin` の場合、`LOAD_TRANSDATA.dat`
- 文字エンコーディング: UTF-8。
- フォーマット: カンマ区切り (CSV)。
    - 1行目: ヘッダー行。COPY句ファイルで定義された各フィールドの「項目名」がカンマ区切りでダブルクォート(")で囲まれて出力されます。
    - 2行目以降: 変換されたデータ。各フィールドの値がカンマ区切りでダブルクォート(")で囲まれて出力されます。
    - フィールド内の特殊文字: フィールド内のダブルクォート(")は二重引用符("")にエスケープされ、フィールド値全体がダブルクォート(")で囲まれます。カンマや改行文字はフィールド値の一部として含まれていても、このダブルクォート囲みにより適切に扱われます。

DATファイル出力例 (LOAD_SAMPLE.dat):

コード スニペット
```
"社員番号","氏名カナ","部署コード","基本給","手当","評価ポイント","備考","レコード種別","更新日"
"A0012345","サンプル　タロウ","0101","+1234567","+12345.00","0.789","これは備考です","1","20250518"
"B0054321","テスト　ハナコ","0203","+2345678","+543.21","0.123","","2","20250517"
...
```

- パック*10*進数 *(COMP-3)* の出力例では、符号が明示的に付与されています（+または-）。

6. 処理フロー

プログラムの実行は、`if __name__ == "__main__":` ブロックから開始されます。

1. 初期設定: 入出力ディレクトリパス、処理対象拡張子などを定数から設定します。
2. 出力ディレクトリ作成: 出力ディレクトリが存在しない場合、自動的に作成します。
3. 文字コード変換マップ読み込み: 標準入力で指定されたパスから文字コード変換定義ファイルを読み込み、変換マップ辞書を生成します。読み込みに失敗した場合、プログラムは終了します。
4. バイナリファイル検索: 入力バイナリディレクトリ (`./iBIN`) から指定拡張子 (`.bin`) のファイルリストを取得します。ファイルが見つからない場合は警告を表示して正常終了します。
5. ファイルごとの処理ループ: 取得したバイナリファイルリストの各ファイルに対して以下の処理を行います。a. ファイルパス生成: バイナリファイル名から、対応するCOPY句ファイル (`./iCPY/CPY_*.txt`) および出力DATファイル (`./oDAT/LOAD_*.dat`) のパスを生成します。b. **Converter** インスタンス生成: `BinaryToDatConverter` クラスのインスタンスを生成します。この際、バイナリファイルパス、COPY句ファイルパス、DATファイルパス、文字コード変換マップ、およびEBCDICエンコーディング設定を渡します。c. 変換処理実行: `converter.process()` メソッドを呼び出し、変換処理を実行します。i. 入力ファイル存在チェック: バイナリファイルとCOPY句ファイルの存在を確認します。見つからない場合はエラーコードを返し、このファイルの処理を終了します。ii. **COPY**句ファイル解析: COPY句ファイルを読み込み、`parse_cpy_field_definitions` 関数でレコード長とフィールド定義リストを解析します。解析エラーがある場合はエラーコードを返し、このファイルの処理を終了します。iii. 出力**DAT**ファイルオープン: 出力DATファイルをUTF-8エンコーディングで書き込みモードでオープンします。iv. ヘッダー行書き込み: フィールド定義リストからヘッダー行を生成し、DATファイルに書き込みます。v. バイナリファイル読み込みループ: バイナリファイルをバイナリ読み込みモードでオープンし、解析したレコード長単位でデータを順次読み込みます。vi. レコード処理: 読み込んだ各レコードのバイト列について、フィールド定義リストに従って各フィールドのバイト列を抽出します。vii. フィールド変換: 抽出した各フィールドのバイト列と定義を `_convert_field` メソッドに渡し、データ型に応じた変換処理を実行します（詳細は「6.1. データ型と変換ルール」参照）。変換中にエラーや警告が発生した場合、`self.errors_encountered` カウンタを増やし、エラー情報を含む文字列を返します。viii. データ行書き込み: 変換後の全フィールド値をCSV形式（ダブルクォート囲み、内部ダブルクォートエスケープ、カンマ区切り）に整形し、DATファイルに1行書き込みます。ix. ループ継続/終了: バイナリファイルの終端に達するか、不完全なレコードを読み込んだ場合はループを終了します。x. 処理結果判定: このファイルの処理中に `self.errors_encountered` が0より大きい場合、エラーコード `PROCESS_ERROR_GENERAL` を返します。それ以外の場合は `PROCESS_SUCCESS` を返します。d. 結果集計: `converter.process()` の戻り値に応じて、成功ファイル数または失敗ファイル数をカウントします。e. ステータス表示: このファイルの処理結果（成功/失敗ステータス）を標準エラー出力に表示します。
6. 終了処理: 全ファイルの処理が完了した後、処理したファイルの総数、成功数、失敗数を表示します。失敗したファイルが一つでもあった場合、プログラムは終了コード 1 で終了します。全て成功した場合は終了コード 0 で終了します。

6.1. データ型と変換ルール

COPY句ファイルの「データ型」列に指定する識別子と、それに対応する変換処理は以下の通り。

データ型 識別子	説明	バイナリ形式 (想定例)	変換処理	出力DAT 形式	備考
`X`	文字列	EBCDIC	`convert_ebcdic_to_string` でASCII互換Unicodeに変換。 NULL文字(0x00)除去、前後の空白除去。	文字列	
`9`	ゾーン10進数 (符号なし)	EBCDIC 例: `F1F2F3` -> "123"	`convert_ebcdic_to_string` で数値文字列に変換。 NULL文字(0x00)除去、前後の空白除去、先行ゼロ除去。全て0なら"0"。 	文字列	
`N`	日本語文字列	JEF	`convert_jef_chars` でJEF変換マップ利用。 JEF拡張文字`@@`を全角スペースに置換、未定義コードを`UNDEFINED_CHAR_REPLACEMENT`に置換、 前後トリム。	文字列	変換マップはJEF 2Byte HEX → Unicode HEX。 2バイト固定処理。
`V9`	先頭小数点のゾーン10進数	EBCDIC 例: `F1F2F3` -> ".123"	`convert_ebcdic_to_string` で数値文字列に変換後、先頭に `"0."` を付加。 前後の空白除去。	文字列	元コードの挙動を維持。

`9(...)V9(...)`
例:
`9(5)V9(2)`
	ゾーン10進数
	(小数点あり)
		EBCDIC	`get_digits_from_pic_type` で小数部桁数取得。
			
			`convert_ebcdic_zoned_decimal` で小数点挿入、前後トリム。
				文字列	

`P9(...)V9(...)`,
`PS9(...)V9(...)`
例:
`P9(7)V9(2)`
	パック10進数
	(小数点あり)
		COMP-3	`get_digits_from_pic_type` で小数部桁数取得。
			`convert_comp3_bytes` で変換 (decimal_places = 小数部桁数)。
				数値
				(文字列化)
					符号付き
					
					不正データはERROR 文字列。

`P9(...)`, `S9(...)`, `PS9(...)`, `SP9(...)` 例: `S9(7)`	パック10進数 (整数)	COMP-3	`convert_comp3_bytes` で変換 (decimal_places = -1)。	数値 (文字列化)	符号付き 不正データはERROR 文字列。
`P9`, `S9`, `PS9`, `SP9`	パック10進数 (整数)	COMP-3	`convert_comp3_bytes` で変換 (decimal_places = -1)。	数値 (文字列化)	符号付き 不正データはERROR 文字列。
`PV9`	パック10進数 (小数部のみ)	COMP-3	COPY句の「数値属性」列から小数点以下の 桁数n取得。 `convert_comp3_bytes` で変換 (decimal_places = n)。	数値 (文字列化)	符号付き 数値属性の指定必須。 不正データはERROR文字列。
`PSV9`	パック10進数 (小数部のみ)	COMP-3	COPY句の「数値属性」列から小数点以下の 桁数n取得。 `convert_comp3_bytes` で変換 (decimal_places = n)。	数値 (文字列化)	符号付き。 数値属性の指定必須。 不正データはERROR文字列。

- 注意:
  - 上記データ型以外の識別子が指定された場合、警告ログが出力され、当該フィールドは `"UNSUPPORTED_TYPE(データ型):バイト列hex"` のように出力されます。
  - 変換処理中に不正なデータが検出された場合、警告ログが出力され、出力されるCSVフィールド値にはエラー情報を含む文字列 （例: `"ERROR(COMP3_INVALID):バイト列hex"`）が設定されます。
  - EBCDIC変換関数 (`convert_ebcdic_to_string`, `convert_ebcdic_zoned_decimal`) は、変換できないEBCDIC文字やASCII範囲外の文字を無視（除去）します。
  - JEF変換関数 (`convert_jef_chars`) は、変換マップに未登録のJEFコードを `UNDEFINED_CHAR_REPLACEMENT` に置き換えます。2バイト固定で処理するため、不完全なバイトシーケンスは未定義として扱われます。

7. 主要なモジュール、クラス、関数

- 定数群: 設定値や固定値を定義します。
- ヘルパー関数:
  - `get_filenames_with_extension(directory_path, extension)`: 指定ディレクトリ・拡張子のファイルリストを取得。
  - `check_file_exists(filepath)`: ファイル/ディレクトリの存在確認。
  - `get_filename_from_path(filepath)`: パスからファイル名（拡張子付き）を取得。
- 設定読み込み関数:
  - `load_conversion_map(file_path)`: 文字コード変換定義ファイルを読み込み、JEF HEX -> Unicode HEX の辞書を作成して返します。
- **COPY**ファイル解析関連:
  - `read_cpy_file(filepath)`: COPY句ファイルを読み込み、行ごとのリストで返します。
  - `parse_cpy_field_definitions(cpy_lines)`: COPY句の行リストを解析し、レコード長とフィールド定義リストを抽出します。
  - `get_digits_from_pic_type(pic_type_str)`: PIC表現文字列 (例: "9(5)V9(2)") から整数部・小数部の桁数を抽出します。
- データ型変換関数:
  - `convert_ebcdic_to_string(ebcdic_bytes, ebcdic_encoding)`: EBCDICバイト列をASCII互換Unicode文字列に変換。
  - `convert_jef_chars(jef_bytes, conversion_map)`: JEFバイト列を変換マップに従ってUnicode文字列に変換。
  - `convert_ebcdic_zoned_decimal(ebcdic_bytes, decimal_digits, ebcdic_encoding)`: ゾーン10進数 (EBCDIC) バイト列を小数点付き数値文字列に変換。
  - `convert_comp3_bytes(comp3_bytes, decimal_places)`: パック10進数 (COMP-3) バイト列を小数点付きまたは整数文字列に変換する汎用関数。
- メイン処理クラス (**`BinaryToDatConverter`**):
  - `__init__(self, binary_filepath, cpy_filepath, dat_filepath, conversion_map, ebcdic_encoding)`: インスタンスの初期化。
  - `process(self)`: 単一のバイナリファイルとCOPY句定義に基づき、DATファイルへの変換処理を行うコアメソッド。ファイルI/O、レコードループ、フィールド抽出を管理します。
  - `_convert_field(self, field_def, field_data)`: 単一のフィールドデータを、定義された型に従って変換する内部メソッド。各データ型変換関数を呼び分けます。
- メイン実行ブロック (**`if __name__ == "__main__":`**): プログラムのエントリポイント。設定読み込み、ディレクトリ準備、ファイル検索、各ファイルに対する `BinaryToDatConverter` インスタンス生成と実行、結果集計、終了コード設定を行います。

8. 設定値

以下の定数は、スクリプトの冒頭部分で設定されています。必要に応じて変更可能です。

- `DEFAULT_EBCDIC_ENCODING`: デフォルトEBCDICエンコーディング名（例: `'cp500'`）
- `JEF_AT_MARK_PAIR_BYTES`: JEF @@ バイトシーケンス（例: `b'\x42\x42'`）
- `FULL_WIDTH_SPACE_UTF8`: JEF @@ の置換先Unicode文字（全角スペース）（例: `'　'`）
- `UNDEFINED_CHAR_REPLACEMENT`: JEF変換マップ未登録文字の置換用文字（例: `'★'`）
- `BINARY_FILE_EXTENSION`: 入力バイナリファイルの拡張子（例: `'bin'`）
- `COPY_FILE_EXTENSION`: 入力COPY句ファイルの拡張子（例: `'txt'`）
- `DAT_FILE_EXTENSION`: 出力DATファイルの拡張子（例: `'dat'`）
- `INPUT_BINARY_DIR`: 入力バイナリファイルが格納されているディレクトリパス（例: `'./iBIN'`）
- `INPUT_CPY_DIR`: 入力COPY句ファイルが格納されているディレクトリパス（例: `'./iCPY'`）
- `OUTPUT_DAT_DIR`: 出力DATファイルを格納するディレクトリパス（例: `'./oDAT'`）
- `CSV_DELIMITER`: CSVの区切り文字（例: `,`）
- `CSV_QUOTECHAR`: CSVの引用符（例: `"`）
- `CSV_ESCAPED_QUOTE`: CSV内で引用符をエスケープする際の方法（例: `""`）

文字コード変換定義ファイルのパスは、プログラム実行時に標準入力から受け取ります。

9. エラーハンドリングとログ

処理中の主要なステップ、成功、警告、エラーは標準エラー出力 (`sys.stderr`) に表示されます。

`BinaryToDatConverter.process()` メソッドの戻り値（`PROCESS_*` 定数）:

- `0` (`PROCESS_SUCCESS`): 正常終了
- `-1` (`PROCESS_ERROR_FILE_NOT_FOUND`): バイナリファイルが見つからない
- `-2` (`PROCESS_ERROR_COPY_NOT_FOUND`): COPY句ファイルが見つからない
- `-3` (`PROCESS_ERROR_COPY_PARSE`): COPY句ファイルの解析エラー
- `-4` (`PROCESS_ERROR_GENERAL`): その他の処理エラー（ファイル内の変換エラー、不完全レコードなど）

COPY句ファイルのフォーマット不正、フィールド定義の矛盾（オフセットやバイト長がレコード長を超える等）、未対応のデータ型などもエラー/警告としてログ出力され、当該ファイルの `self.errors_encountered` カウンタが増加します。

最終的に、全ファイルの処理が完了した後、失敗ファイル数（`BinaryToDatConverter.process()` が `PROCESS_SUCCESS` 以外を返したファイル数）が0より大きい場合、プログラムの終了コードとして 1 を返し、全て正常終了した場合は 0 を返します。

10. 実行方法

- Python 3.7 以降の実行環境が必要です。
- 本スクリプトファイル (例: `cnvCP500EUCJP2UTF8.py`) を任意の場所に配置します。

スクリプトファイルと同じ階層に、以下のディレクトリ構造を作成し、入出力ファイルを配置します。

<アクセス権のある任意のフォルダ>

```
        │
        ├── cnvCP500EUCJP2UTF8.py  (本スクリプト)
        ├── iBIN/
        │       ├── data1.bin
        │       └── data2.bin
        ├── iCPY/
        │       ├── CPY_data1.txt
        │       └── CPY_data2.txt
        └── (文字コード変換定義ファイルは任意の場所)
```

コンソールから以下のコマンドを実行します。
実行時に、文字コード変換定義ファイルのパス入力を求められます。

Bash

```
python3  cnvCP500EUCJP2UTF8.py
```

- 処理が完了すると、`./oDAT` ディレクトリ (存在しない場合は自動作成) 内に変換後のDATファイルが生成されます。

11. 前提条件・制約事項

- **Python**バージョン: Python 3.7 以上推奨。
- 外部ライブラリ: 標準ライブラリのみ使用 (`os`, `sys`, `glob`, `re`, `struct`, `typing`)。
- **COPY**句ファイルの文字エンコーディング: UTF-8 を前提としています。
- バイナリファイルのエンコーディング: EBCDIC (cp500等のデフォルト設定) および JEF を主に想定しています。他のEBCDICコードページを使用する場合は、`DEFAULT_EBCDIC_ENCODING` 定数の変更や、必要に応じて追加の文字コード変換マップの定義が必要になる場合があります。
- **JEF**変換: `convert_jef_chars` 関数は、入力バイト列を2バイトずつ区切り、変換マップを適用する2バイト固定長の変換を想定しています。JEFデータ中に1バイト文字や、2バイト固定では扱えないシーケンスが含まれる場合、正しく処理できない可能性があります。
- パフォーマンス: 非常に巨大なファイル (数GB以上) を処理する場合、ファイルI/Oや変換処理の時間は増加します。現状はレコード単位での逐次処理のため、メモリ使用量は比較的抑えられます。
- **CSV**エスケープ: 出力DATファイル (CSV) のフィールド値については、ダブルクォート(`"`)を二重引用符(`""`)にエスケープし、フィールド値全体をダブルクォート(`"`)で囲む処理を行っています。これにより、フィールド値内にカンマ(`,`)や改行文字が含まれていても、CSVとして正しく解釈されるようになります。