

# Documentation - DQMT File Formats

Author: Safi Ebeid

Last Updated: 2/1/2025

This documentation is meant to serve as a reference for the proprietary binary file formats used by DQMT (*Dragon Quest Monsters Terry's Wonderland 3D*), and to preserve my efforts in reversing them. Note that this doc is still a work in progress. The game uses many custom file formats and the ones listed here represent only a fraction of them, which I have reversed.

## Table of Contents

1. SkillTbl.bin.....	pg. 2
2. EnemyKindTbl.bin.....	pg. 4
3. LevelUpTbl.bin.....	pg. 7
4. CombinationKindTbl.bin.....	pg. 8
5. Combination4GTbl.bin.....	pg. 9
6. SkillSpEvoTbl.bin.....	pg. 10
7. SkillPointTbl.bin.....	pg. 11
8. .BINJ.....	pg. 12
9. .BCLIM.....	pg. 13

# SkillTbl.bin

=====  
Description: Contains a table in which each entry contains information about a skill, such as skill points needed to unlock a move, move IDs, and trait IDs (ones that are unlockable through a skill).

## Sections

=====  
The sections of the file in the order that they appear:

1. SktHeader
2. SktEntry
  - a. SktMoveSubEntry

**NOTE:** all values are little-endian

### SktHeader (0x8 (8) bytes)

=====  
Description: general information

```
typedef struct
{
    UINT32 magic;           //"SKIL"
    UINT32 num_entries;     //number of skill entries
}SktHeader;
```

### SktEntry (0x6E (110) bytes)

=====  
Description: contains unlockable moveset/traits as well as the amount of SP needed to unlock each move/trait.

```
typedef struct __attribute__((packed, aligned(1)))
{
    UINT8 req_sp[10];      //Skill points required to unlock each move
    SktMoveSubEntry moves[10]; //each move has its own SktMoveSubEntry
    UINT16 traits[10];     //Trait ID if skill has trait
}SktEntry;
```

**SktMoveSubEntry (0x8 (8) bytes)**

=====

Description: contains move ID (among unknown/padding values)

```
typedef struct __attribute__((packed, aligned(1)))  
{  
    UINT16 id;    //move ID  
    UINT16 unknown;  
    UINT32 padding;  
}SktMoveSubEntry;
```

# EnmyKindTbl.bin

=====  
Description: Contains the max stats data for every monster in the game

## Sections

=====  
The sections of the file in the order that they appear:

1. EktHeader
2. EktEntry

**NOTE:** all values are in little-endian

## EktHeader (0x8 (8) bytes)

=====  
Description: general information about the file

```
typedef struct {
    UINT32 magic;          //"EKT\x00"
    UINT32 num_entries;    // # of entries (should be 0x400, or 1024)
} EktHeader;
```

## EktEntry (0x68 (104) bytes)

=====  
Description: comprehensive stats for the monster

```
typedef struct __attribute__((packed, aligned(1))) {
    UINT8 padding_0[0x6];
    UINT8 species_and_rank; // "keitou" and "rank" (1)
    UINT8 size;             // size of monster (2)
    UINT16 rank_number;     // rank NUMBER (3)
    UINT16 monster_id;      // internal monster ID (4)
    UINT8 padding_1[0x4];
    UINT16 max_hp;          // Max HP
    UINT16 max_mp;          // Max MP
    UINT16 max_atk;         // Max ATK
    UINT16 max_def;         // Max DEF
    UINT16 max_spd;         // Max SPD
    UINT16 max_int;         // Max INT
    UINT8 padding_2[0xC];
    UINT8 resistance[0xE];  // resistance to move types (5)
    UINT16 equippable_weapons; // weapons (6)
    UINT16 traits[0x6];     // trait IDs (7)
    UINT8 padding_3[0x4];
}
```

```

UINT8 levelup_table;      //Level Up Table ID (8)
UINT8 padding_4[0x1A];
UINT16 skill;             //fixed skill ID
UINT8 padding_5[0x2];
}EktEntry;

```

#### Annotations:

- (1): upper nibble: "keitou" (slime=1, dragon=2, nature, beast, ..., ???=8)  
lower nibble: Rank (F=1, E=2, ..., SS=8, SS+Star=9)
- (2): lower nibble: Size (see table below)

Value	Size Trait
0	Small
1	Standard
2	Standard
3	Mega
4	Mega
5	Mega
6	Giga
7	Giga
8	Giga

- (3): Rank (not like E, A, SS, but the rank NUMBER)
- (4): Monster ID (Only for boss monsters 0x320-0x344)  
- normal monsters do not have IDs in their entries, because the monster ID is just the entry # in the table (this file)
- (5): Resistance (Length: 14 bytes)  
- each nibble represents a spell/attack type (eg. "mera", "bagi")  
- 28 total resistance types  
- possible values range from 0-7:  
- ("yowai", "futsuu", "keigen", "hangen", "gekigen", "mukou", "kaifuku", "hansha")  
- 0xB means it will increase as the monster levels up
- (6): Equippable Weapons  
- bits: **0cCwhasS**  
- In order, from left to right: (cane (bit 6), Claws, whip, hammer, axe, spear, Sword (bit 0))  
- 1 means the weapon can be equipped by the monster, 0 means no
- (7): Trait IDs; each monster has 6 entries but the max number of traits is dependent on the size of the monster (i.e. small/standard monsters can only have up to 4 traits, and only Giga monsters can have 6 traits excluding size trait). The first entry in the array is the first trait.

**(8):** This value is the ID of the level-up table used for that monster. The table can be found in *LevelUpTbl.bin*.

# LevelUpTbl.bin

=====  
Description: Contains a tables of XP needed to advance to the each level

## Sections

=====  
The sections of the file in the order that they appear:

1. Table Entry

**NOTE:** all values are in little-endian

## Table Entry (0x190 (400) bytes)

=====  
Description: contains 100 XP values, each corresponding to how much is needed to advance to the next level

```
typedef struct {  
    UINT32 table[100];      //each word is a XP value  
}LevelUpTableEntry;
```

Note: The first table is empty. (Data starts at offset 0x190)

# CombinationKindTbl.bin

=====  
Description: contains a table of monster fusion combinations for those who only require two parents. There are 371 entries.

## Sections

=====  
The sections of the file in the order that they appear:

1. Table Entry

**NOTE:** all values are in little-endian

## Table Entry (0x6 (6) bytes)

=====  
Description: each entry contains the child ID and parent IDs

```
typedef struct {  
    UINT16 child_monster_id;      //fusion product  
    UINT16 parent_1_monster_id;   //first parent  
    UINT16 parent_2_monster_id;   //second parent  
}FusionTableEntry;
```



# Combination4GTbl.bin

=====

Description: contains a table of monster fusion combinations for those who require four parents. There are 70 entries.

## Sections

=====

The sections of the file in the order that they appear:

1. Table Entry

**NOTE:** all values are in little-endian

## Table Entry (0xA (10) bytes)

=====

Description: each entry contains the child ID and parent IDs

```
typedef struct {
    UINT16 child_monster_id;      //fusion product
    UINT16 parent_1_monster_id;  //first parent
    UINT16 parent_2_monster_id;  //second parent
    UINT16 parent_3_monster_id;  //third parent
    UINT16 parent_4_monster_id;  //fourth parent
}FusionTableEntry;
```

# SkillSpEvoTbl.bin

=====

Description: contains a table of skill combinations that form SP skills

## Sections

=====

The sections of the file in the order that they appear:

1. Header
2. Table Entry

**NOTE:** all values are in little-endian

## Header (0x8 (8) bytes)

=====

Description: each entry contains the child ID and parent IDs

```
typedef struct {
    UINT8  signature[4];           //"SESP"
    UINT32 num_entries;           //# of entries in the file
}SkillSpEvoHeader;
```

## Table Entry (0x1C (28) bytes)

=====

Description: each entry contains the IDs of the child skill and parent skills as well as the skill points needed for each parent skill

```
typedef struct {
    UINT16 perent_skills[6];       //parent skill IDs
    UINT16 parent_sps[6];         //parent skill points needed
    UINT16 sp_skill_id;          //SP skill ID
    UINT16 padding;
}SkillSpEvoTableEntry;
```

# SkillPointTbl.bin

=====  
Description: contains a table of skill points earned for each level. There are 0x64 (100) entries.

## Sections

=====  
The sections of the file in the order that they appear:

1. Table

## Table

=====  
Description: each byte tells how much SP earned at that level

```
typedef struct {  
    UINT8 sp_earned[100];           //skill points earned for each level  
}SkillPointTable;
```

**Important:** *The file format/structure of .binj and .bclim files have been known to the community for a while now, however since there was a lack of documentation for those files I have chosen to include them here for reference. These file formats were not entirely reversed by me.*

## .BINJ

=====

Description: contains a table of messages in japanese.

### Sections

=====

The sections of the file in the order that they appear:

1. Initial Block
2. Header
3. Pointer Blocks
4. Data Blocks

#### Initial Block (0x4 (4) bytes)

=====

0x00[4]: header size (we'll call this value "N")

#### Header (4 \* N bytes)

=====

[(0 to (N-1)) \* 4]: # of entries

- The sum of these values is the total number of entries in the file
- We'll call this value "T"

**NOTE:** The max value allowed is 0x400. If there are more entries, simply add the subsequent values in this block to get the total # of entries

Ex. if there are 1143 entries and "N" = 4, then the first word will be 0x400 and the following words will be smaller numbers that add up to 1143

#### Pointer Block (4 bytes)

=====

Description: Contains pointer (from file origin) to corresponding raw text entry in the file

0x00[4]: pointer to text entry

#### Data Block (variable size)

=====

Description: Contains raw text data with special characters that require a special table in order to decode. The entry stops at the terminator character [0xE31B]. Each character can be up to 3 bytes.

0x00[variable]: raw text data

# **.BCLIM (Work in Progress)**

=====

Description: Image file format for 3ds

## **Sections**

=====

The sections of the file in the order that they appear:

1. Image Block(s)
2. BCLIM Header
3. Image Header

**NOTE:** all values were observed to be in little-endian

## **BCLIM Header (0x14 (20) bytes)**

=====

Description: contains general information about the file

```
typedef struct __attribute__((packed, aligned(1)))
{
    UINT8 signature[4];          //should say "CLIM"
    UINT16 unknown_1;           //usually 0xFFFE (endianness?)
    UINT32 image_header_size;    //size of image header
    UINT8 unknown_2[2];
    UINT32 file_size;           //length of entire file
    UINT32 unknown_3;
}BclimHeader;
```

## **Image Header (0x14 (20) bytes)**

=====

Description: contains basic information about the image such as dimensions, format, and size

```
typedef struct __attribute__((packed, aligned(1)))
{
    UINT8 signature[4];          //"imag"
    UINT32 header_size;         //length of header following the signature
    UINT16 width;               //width of image in pixels
    UINT16 height;              //height of image in pixels
    UINT32 format;              //encoding format (1)
    UINT32 image_size;          //
}ImageHeader;
```

Annotations:

(1): known values: 8 = RGBA4444

Example:

```
//each field is 4 bits
//total: 2 bytes
typedef struct __attribute__((packed, aligned(1)))
{
    UINT8 R : 4;
    UINT8 G : 4;
    UINT8 B : 4;
    UINT8 A : 4;
}RGBA4444;
```

## Image Block (? bytes)

=====

Description: contains raw pixel data in specified format. Size of each block depends on the encoding format. For RGBA4444, the size is 2 bytes since each field is 4 bits.

```
//align to 8-byte boundary
//ex. align(13) -> 16, align(28) -> 32
UINT32 BclimParser::align(UINT32 n)
{
    UINT32 s = 8;
    while(s < n)
    {
        s *= 2;
    }
    return s;
}
```