IRD File Format

Compression

An IRD file is compressed using GZip compression. It's important that no additional metadata is specified, such as comments, filenames, etc. The upload script cannot handle those. As such, the GZip format must follow the following requirements.

A compressed file starts with a 2 byte magic (0x1f, 0x8b). After that, the compression method (one byte, value 8) is specified, followed by a flags byte (value 0). Then 4 bytes which defined the modification time as Unix epoch, followed by extra flags (one byte, value 0). This should give you a header of 10 bytes. Again, filename or comment is not allowed in this header.

After the header, the gzip inflated content is added. Finally, 8 bytes follow, 4 bytes for the crc of the content, then 4 bytes which define the size of the file.

Content specification

The content of the GZipped file is the actual IRD file. The specification of the IRD file follow below.

Magic

Every IRD file starts with 4 bytes, indicating the magic. The magic should read "3IRD".

```
0000000 33 49 52 44 06 42 4c 45 53 30 30 35 31 36 0c 41 3IRL.BLES00516.A
```

Version

After the magic, one byte follows which defines the version of the IRD file. The current version is 6, lower versions are no longer supported as they lack certain information about the original ISO. Think about application version, release version, and the game id. Apparently, the game id defined in the header of the BluRay can differ from the one specified in the PARAM.SFO file. From version 5 and onwards, the PARAM.SFO is parsed, and some values are included in the IRD file.

```
00000000 33 49 52 44 06 42 4c 45 53 30 30 35 31 36 0c 41 3IRD.BLES00516.A
```

Game ID

The game id is specified without any dashes, which means it takes up 9 bytes. It shouldn't be encoded, but just ASCII coded (single byte characters). In the example below the Game ID is BLES00516.

```
00000000 33 49 52 44 06 42 4c 45 53 30 30 35 31 36 0c 41 3IRD.BLES00516.A
```

Game name

After the game id, the name of the game is included. This name is prefixed with a single byte which specifies the length of the name. In the example below, the length is 0x0c, the name is AFRO SAMURAI.

```
00000000 33 49 52 44 06 42 4c 45 53 30 30 35 31 36 0c 41 3IRD.BLES00516.A 00000010 46 52 4f 20 53 41 4d 55 52 41 49 32 2e 36 30 30 FRO SAMURAI 2.600
```

Update version

What follows, is the version of the PS3UPDAT.PUP file. Most games have such a file, which contains the latest version of the update at the moment the disc was released. Its length is four bytes, and it's written in plain text. In case the disc doesn't have an update, the version is 0000 (in text!). In the example below, the version is 2.60

```
00000010 46 52 4f 20 53 41 4d 55 52 41 49 32 2e 36 30 30 FRO SAMURAI 2.600
```

Game version

Next is the game version, or release version. Whenever a special edition is released, the game id stays the same (sometimes) but the release version is changed. This value is extracted from the PARAM.SFO file, and it's extracted from the VERSION field. The length of the field is 5 bytes, and it's written in plain text. In the example below, the version is 01.00

App version

The app version is the version of the application or patch. This value also comes from PARAM.SFO, the APP_VER field. The length is also 5 bytes, and written in plain text. In the example below, the version is 01.00

Header and footer

Next up are the header and footer of the game disc. To determine the header of the original disc, find the sector where the first file starts. The header contains everything from sector 0 to the start of the first file. The footer contains everything from the end of the last region until the end of the file. The last region always contains the PS3UPDAT.PUP file, and its length is always 256 MB. The footer is usually somewhere around 32 sectors, and contains almost only zeroes. The header length depends on the amount of regions, and the amount of files. By default the first file starts at sector 0x448. The first file is always PS3_DISC.SFB. Header and footer are also GZipped, before added to the IRD file. They are prefixed with 4 bytes, defining the length (uint).

Region hashes

For every region the Ripp3r application processes, the md5 hash is calculated. The next part of the file contains the md5 hash for every complete region in the file. There are two exceptions for this calculation. The first region starts it's calculation at the start sector of the first file. The last regions ends with the end of the last file (PS3_UPDAT.PUP).

When the hashes are written to the IRD file, first one byte with the amount of hashes is written. Then the hashes itself follow (16 bytes each).

File hashes

Every file on the original disc is decrypted, and the md5 hash is calculated. This allows us to alert the user when building an iso which file is invalid. The file hashes are stored in some kind of list, by saving the start sector and hash. First, a 4 byte integer is written containing the amount of files. Next, for every file on the disc, 8 bytes are written containing the start sector of the file, followed by the md5 hash (16 bytes).

Extra attributes

For future developments, two shorts are written to the IRD file. The first one is meant for extra configuration values, the second one for file attachments. At the moment, both must be filled with zero, and are ignored.

Authorization data

The original iso contains plain and encrypted content. To be able to successfully re-encrypt the plain files, we need the original keys from the disc. First, the D1 key is written (16 bytes long). Next up, the D2 key is written (also 16 bytes long). Finally, the PIC data is written (115 bytes long).

A special note about the D2 key is in order. This key is a unique fingerprint, which is different on each disc produced. Therefore, we need to change this key to hide the origin. The origin *could* lead to a PSN account. As a result, the D2 key is decrypted. The first 12 bytes are copied, but the last 4 bytes are changed. If the last 4 bytes are all zeroes, then no change will be made. If they contain a different value, then it will be changed to 1 (0x00 0x00 0x00 0x01). Then the key is encrypted again, and stored in the IRD file.

When building an ISO file, the D2 is decrypted once again, and the last four bytes are checked for a value other than 0. If it finds that, it will supply a random value between 0 and 0x1FFFFF.

Crc

Finally, the crc value of the content up until now is calculated and added to the file (4 bytes integer).