What is this "re-linking redundant files" about?

by TuxTheWise

I usually write this sentence in my readme files, but recently I've been receiving some questions and tutorial requests about this theme. The problem is that some people that wrote me made some weird questions, showing that they don't know exactly what it's about. This short guide will explain what "re-linking redundant files" is, at least the basic. How to do it will be explained in a (possible) future guide.

Note that the principle of this technique can be applied in any not-dynamic media (like DVDs), so it's not a Dreamcast-only stuff. It also can be used to file formats with structure similar to ISO format.

What are redundant files?

What I call "redundant files" are files with the same content, meaning that they match on a byte-to-byte comparison.

Why are there redundant files?

Imagine you're developing a game. This game has two levels. In the disc, you create a folder LEVEL1 and a folder LEVEL2. Now imagine that you're using a texture BRICKS.BMP (size = 1mb) for a wall, and this texture is used in both levels. You'll have a BRICKS.BMP in LEVEL1 folder, and a BRICKS.BMP in LEVEL2 folder. Note that both files have the exact same information, they're byte-to-byte equal. But since you have two copies of it in the same disc, you need to use 2mb, and 1mb is just the copy of an already existent file.

That was just an example of how redundant files may appear. Note that not necessarily redundant files have the same name.

It's common for Dreamcast games to have 1~10mb of redundant files, but there are extremely cases like Resident Evil 2 where you have a 200mb duplicated file. If you re-link this file like I did in the DCRES release, you can make the game fit on a CD-R without downsampling any video. This technique has a wonderful potential.

The basic ISO structure:

Basically the ISO is the file that will be recorded in the disc. It contains all the "real" data of the disc (physically there are more information), so it contains all the disc files.

Wait... but an ISO is a single file, so how do the reader (we're upper the hardware layer here) know how to find the files? Because ISO has a header. In this header, among other information, you'll find the logic position and the size of all the files inside the ISO, so the software knows exactly where to read to retrieve a file.

OK, let's invent a structure like an ISO, but instead of bytes, we'll be using

characters. We'll have three files (files A, B and C) with the following content:

```
A: TUX
B: DCRES
C: TUX
```

Our task is to put everything together in a single file, that from now on we'll call IZO. So a natural way for an IZO would be to put everything together:

```
TUXDCRESTUX
```

Okay, everything is in a single file. But there is a problem: we can't retrieve the information! If you haven't seen the original files yet, and I tell you to tell me what the second file of the IZO is, you won't be able to tell me! Would it be DCRES? But it can also be XDCREST!

However, you could retrieve the information if I told you that the second file starts at position 4 and is 5 characters long. So we need to invent some kind of header.

For commodity, we'll also put the file name in the header.

For the IZO file, we could invent this type of header:

Quote:

- IZO header starts with the number of files (two characters). After that you have the file list, a file entry after another.
- Each entry will have 5 characters.
- The first character is the file name.
- The next two characters are the start position of the file.
- The next two characters are the size of the file. So let's write our header!

We have three files, so we start with:

```
03
X XX XX
X XX XX
X XX XX
TUXDCRESTUX
```

First file is file A, that starts at position 18 and is 3 characters long, so we would get:

```
03
A 18 03
X XX XX
X XX XX
TUXDCRESTUX
```

And we write the same information for files B and C:

```
03
A 18 03
B 21 05
```

```
C 26 03
TUXDCRESTUX
```

So our final file would be:

```
03A1803B2105C2603TUXDCRESTUX
```

If I only give you this sequence of characters, you won't be able to tell me a thing about it. Now if I give you the IZO specifications, you'll be able to tell me exactly what files A, B and C are.

ISO has the same logic for files location, and it's important that you understand the example above so you can see where I want to get.

Re-linking redundant files:
In the fictional IZO case above, you probably noticed that the files A and C have the same content: TUX. You may be wondering... could we make the message three character shorter? In other words, can I save the space of this redundant file? Yes, you can. The idea is to make the information of the header of both files A and C to point to the same data. Simple, right?

This is a simulation of how it's done in the Dreamcast games.

I have files A, B and C in my directory. Somehow, I know A and C are redundant files. So I'll erase file C and put a file with 0 characters in its place. So I have:

```
A: TUX
B: DCRES
C: (nothing)
```

Now I'll create an IZO for this set of files:

```
03
A 18 03
B 21 05
C -- 00
TUXDCRES
```

The "--" can be any valid position, because file C now is 0 bytes long.

But I know file A and C originally had the same data, right? And I also know that I replaced the file C for an empty file. So now I'll copy A's information over C's:

```
03
A 18 03
B 21 05
C 18 03
TUXDCRES
```

Make the test now, try to identify files A, B and C. You'll see that you'll have the exact information that we started from:

```
A: TUX
```

```
B: DCRES
C: TUX
```

Applications will see files A and C as different files, but they'll be read in the same position in the IZO file. If we compare the IZO with redundant information and the IZO without redundant information (after the re-link) we'll get:

`03A1803B2105C2603TUXDCRESTUX` (28 characters)
`03A1803B2105C1803TUXDCRES` (25 characters)

Our IZO got 3 characters smaller without loss of information. That's the idea of redundant files re-linking, that's how you save space with it. You just have to imagine an ISO as a set of bytes (not characters) with its own logic of files positioning (that has the same logic of our fictional IZO).

Final notes:
As said, the operational part will be in a possible future guide. But know that you'll have to fully understand this guide if you want to go on with this.