

system.res

god\_menu.res

SYSTEM\_UPDATE.EDAT

877\_Step\_DlcSurvival051.res

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	50	72	65	73	20	00	00	00	08	02	6E	7C	03	00	00	00	Pres .....n .....
00000010	70	23	00	00	00	00	00	00	00	00	00	00	00	00	00	00	p#.....
00000020	80	0E	00	00	03	00	00	00	60	0E	00	00	01	00	00	00	€.....`.....
00000030	60	00	00	00	1F	00	00	00	00	00	00	00	00	00	00	00	`.....
00000040	40	04	00	00	47	00	00	00	20	0D	00	00	0A	00	00	00	@...G... ..
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

[Start Point]

- ~ 0x00: Magic/Header: 4 bytes (Pres)
- ~ 0x04: 4 bytes, little endian offset (0x20). jumps to table offset
- ~ 0x08: byte, group table count. (always set to 08)
- ~ 0x09: some kind of value?. changes on different RES archives.
- ~ 0x0A: 2 bytes checksum?. not sure what to do with it. (but i guess it could be a a MD5 hash due to libmd5.prx being used)
- ~ 0x0C: 4 bytes, seems to be stuck at 3, must be a version of that RES archive?
- ~ 0x10: 4 bytes little endian offset, jumps you to where certain chunks of data are stored (related to ToC)
- ~ 0x14->0x1F: lengthy zeroes, 4 bytes each. not really useful. but can be a table offset out of nowhere so keep an eye out.

[Table Offsets] ranging: 0x20-0x5F

- ~ 0x20: empty usually on some other .res archive, but it represent as ToC offset (`00 15 00 00` -> 0x1500 for example)
- ~ 0x24: empty usually on some other .res archive, but it represent as count for ToC (`03 00 00 00` -> 3 for example)
  - rinse and repeat for others like 0x28 as ToC offset and 0x2C as ToC count.
  - Table Offsets always uses 16 bytes. be creative when reading them.

[Table of Contents] (ToC) [32 bytes in size]

~ **File Offset**: a little endian offset. brings you to a area where a offset chunk is stored.

~ **Size/Compressed Size**: defines the size of that area starting with

**File Offset**. **Size** can vary if it's **ZLIB** compressed or not based on it's header (blz2)

~ **Name Offset Table**: a little endian offset. brings you to a area where the name is located.

~ **Name Element**: a name element (that's how i call it). can vary from 1, 2, 3.

~ **Zeroes**: bunch of zeroes, length of 12 (0xC)

~ **Size (Decompressed)**: Game's expected decompressed size of that **Compressed Size**. Value can match with **Size** if it's not compressed by **ZLIB**.

[Identifying External File Data]

in **0x940**. you see that the 4th byte has **`40`**. let's call this

**Address Types**. If you encounter any of these values, it means that the file source is external. here's the following lists for these values

~ 40 (0x4) = package.rdp

~ 50 (0x5) = data.rdp

~ 60 (0x6) = patch.rdp

~ C0 (0xC) = Current (no need to go to a external source)

~ D0 (0xD) = Current (no need to go to a external source)

If you encounter any of these. You will need to do multiplications.

000007C0	10 12 00 C0	01 00 00 00	20 12 00 00	01 00 00 00	...À.....
000007D0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000007E0	40 12 00 C0	0F 00 00 00	80 12 00 00	01 00 00 00	@..À....€.....
000007F0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000800	A0 12 00 C0	01 00 00 00	B0 12 00 00	01 00 00 00	..À....°.....
00000810	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000820	D0 12 00 C0	01 00 00 00	E0 12 00 00	01 00 00 00	Đ..À....à.....
00000830	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000840	00 13 00 C0	03 00 00 00	10 13 00 00	01 00 00 00	...À.....
00000850	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000860	30 13 00 C0	08 00 00 00	50 13 00 00	01 00 00 00	0..À....P.....
00000870	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000880	80 13 00 C0	08 00 00 00	A0 13 00 00	01 00 00 00	€..À.....
00000890	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000008A0	D0 13 00 C0	08 00 00 00	F0 13 00 00	01 00 00 00	Đ..À....ð.....
000008B0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000008C0	00 00 00 00	00 00 00 00	20 14 00 00	01 00 00 00	.....
000008D0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000008E0	40 14 00 C0	A8 00 00 00	F0 14 00 00	01 00 00 00	@..À`...ð.....
000008F0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000900	10 15 00 C0	12 00 00 00	30 15 00 00	01 00 00 00	...À....0.....
00000910	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000920	50 15 00 C0	02 00 00 00	60 15 00 00	01 00 00 00	P..À....`.....
00000930	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00000940	F5 78 05 40	E4 7E 19 00	70 15 00 00	03 00 00 00	õx.@ä~..p.....
00000950	00 00 00 00	00 00 00 00	00 00 00 00	E4 7E 19 00	.....ä~..

Obtaining **Absolute Offset** (for external files)

~ Reassign **0x940**'s **File Offset** into Big Endian order, you'll get something like this:

“400578F5”/”0x400578F5”.

~ Remove or Change “4” to “0” and multiply the new offset by 800:

“400578F5” \* “800” = “2BC7A800”/”0x2BC7A800”

~ The **true/absolute offset** is “2BC7A800”

**\*.rtbl** files have this structures but spreads out of nowhere with lengthy paddings  
keep an eye out.

# [Name Tables]

Note: If **Name Element** is less than three (sometimes it could be 1). it will rely only on **Name Offset**.

~ **Name Offset**: can be treated as UINT16 or UINT32, this is an offset that jumps you to the File Name.

~ **Extension Offset**: can be treated as UINT16 or UINT32. this offset jumps you to the File Extension.

~ **Padding?**: an offset that jumps you to the zeroes. Very... useless?

00000960	6C 09 00 00	70 09 00 00	73 09 00 00	63 6F 6C 00	1...p...s...col.
00000970	6D 6C 00 00	00 00 00 00	00 00 00 00	00 00 00 00	m1.....
00000980	8C 09 00 00	90 09 00 00	95 09 00 00	4E 50 43 00	E.....*...NPC.
00000990	66 70 74 68	00 00 00 00	00 00 00 00	00 00 00 00	fpth.....
000009A0	AC 09 00 00	BC 09 00 00	C0 09 00 00	4D 53 5F 31	¬...4...À...MS_1
000009B0	30 36 4E 5F	62 6F 78 66	69 73 68 00	72 65 73 00	06N_boxfish.res.
000009C0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
000009D0	DC 09 00 00	EA 09 00 00	EE 09 00 00	4D 53 5F 30	Û...ê...î...MS_0
000009E0	31 30 43 5F	77 68 69 74	65 00 72 65	73 00 00 00	10C_white.res...
000009F0	FC 09 00 00	0E 0A 00 00	12 0A 00 00	65 6E 65 6D	ü.....enem
00000A00	79 5F 65 6E	74 72 79 5F	74 61 62 6C	65 00 74 72	y_entry_table.tr
00000A10	32 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	2.....

# [ZLIB Compression]

0x00: Header 4 bytes. "blz2"

0x04: Compressed file size, 2 bytes.

the rest: compressed data.

Padding: adjustment for another file chunk or compressed data

## [decompression sample code (python)]

```
def blz_decompress(data, csize, dsize):
```

```
data = BytesIO(data)
```

```
magic = data.read(4)
```

```
if magic != b"blz2":
```

```
raise ValueError("Data is not in BLZ2 format.")
```

```
decom = b""
```

```
if dsize >= 0xFFFF:
```

```
size = int.from_bytes(data.read(2), "little")
```

```
ekor = zlib.decompress(data.read(size), -15)
```

```
while data.tell() < csize:
```

```
size = int.from_bytes(data.read(2), "little")
```

```
decom += zlib.decompress(data.read(size), -15)
```

```
return decom + ekor
```

```
else:
```

```
size = int.from_bytes(data.read(2), "little")
```

```
decom = zlib.decompress(data.read(size), -15)
```

```
return decom
```

62	6C	7A	32	7E	00	2B	2E	2E	4C	11	60	60	60	60	62	blz2~.+..L.``b
90	63	60	07	D2	2C	4C	40	82	81	11	88	FF	FF	67	18	.c`.Ò,L@,..^ÿÿg.
04	40	92	09	C6	82	B8	87	03	08	D1	81	01	D0	B9	09	.@'.Æ,+.Ñ..Ð¹.
40	3C	81	11	55	FC	00	23	6E	73	5F	00	E5	3E	00	F1	@<..Uü.#ns_.å>.ñ
0F	74	35	4C	18	4A	99	98	98	A0	41	02	06	05	4E	D8	.t5L.J™~~_Ä...NØ
8D	C5	25	CE	41	A2	78	83	3D	2E	67	33	62	15	63	64	.Å%ÎAçxf=.g3b.cd
C0	A1	FE	3F	1A	06	82	07	0E	D8	E8	E0	CA	E2	92	D4	À;p?..,..øèàÊâ'Ô
DC	78	B7	C4	94	D4	78	A7	9C	C4	E4	EC	78	CF	3C	06	Üx·Ä"Ôx\$œÄäixİ<.
5D	88	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	j^.....