# PBO File Format

From Bohemia Interactive Community

PBO File Format

Disclaimer: This page describes internal undocumented structures of Bohemia Interactive software.

This page contains unofficial information.

Some usage of this information may constitute a violation of the rights of Bohemia Interactive and is in no way endorsed or recommended by Bohemia Interactive.

Bohemia Interactive is not willing to tolerate use of such tools if it contravenes any general licenses granted to end users of this community wiki or BI products.

## Pbo file structure and packing method

### Contents


- 1 Introduction
  - 1.1 Legend
  - 1.2 Compression
  - 1.3 Binarised raP
- 2 Main Format
- 3 Pbo Header Entry
  - 3.1 Null Entries
  - 3.2 HeaderExtension
    - 3.2.1 Resistance Pbo
  - 3.3 OFP Elite Pbo
  - 3.4 ArmA Pbo
- 4 Data compression
- 5 Bibliography


## Introduction

A pbo file originally meant 'packed bank of files'. The acronym can be read as "Packed Bohemia Object". Through use it has come to represent a single 'package' to achieve a result, such as a mission, or an addon.

A .pbo file is the output produced by the Mission Editor when 'exporting' and contains nothing more (and nothing less) than the content of all the files and folders making up a mission or campaign, or addon. It is a single file representation of a folder tree. The key to grasp is that anything you uniquely make in a folder, such as a mission, such as a campaign, such as an addon, can be conveniently packaged into a single file, called, a pbo.

The engine will internally expand any pbo back out to it's original, tree-folder, form.

Additionally, the engine will work with the equivalent non pbo versions of missions or campaigns, but not (unfortunately) Addons. Addons must be in 'pbo format' to be usable by the engine.

### Legend

see Generic FileFormat Data Types

### Compression

In addition to simply packaging all files and folders in a tree into a single file, some, all, or none of the files within can be compressed. Which type of files are compressed is *entirely* optional. Tools for creating compressed pbo files are Makepbo by Amalfi among others. The intent behind compression was for internet use and, in the 'good old days', simply to reduce hard disk storage requirements. The *actual* use of compression (a mild form of run length encoding) is becoming less 'popular' as it does represent a load on the engine. Neither Operation Flashpoint Elite, nor ArmA, can work with *compressed pbo* files. See Elite Pbo's

### Binarised raP

Config.bin definately, and often config.cpp and/or Mission.sqm are stored in binary form. This has no relation to pbo structure, though, and and it is not part of the pbo compression/decompression algorithm. The data for mission.sqm may indeed, also be compressed within the pbo, but the resulting output is often, a raPified version of the original mission.sqm text. It must be further decoded by utilities such as bin2cpp and cpp2bin.

## Main Format

The format of a pbo is extremely simple. It contains

1. a header consisting of contiguous file name stuctures.
2. one, contiguous data block.
3. a checksum (elite) or a signature key (ArmA)

The header defines each file contained in the pbo, its size, date, name, whether it's compressed, and where it 'is' in the following data block. Every file, even zero length ones, are recorded in the header and each is referred to as an 'entry'. Entries, and consequently the 'file' they refer to, are contiguous.

However, note that there is no provision for, and no ability to, store empty folders. Folders as such are indicated simply by being part of the filename. There are no, folder entries, and consequently, empty folders, cannot be included in a pbo because there is no filename associated with them. Put another way, an empty folder, if it could be stored (and it can't), would appear to be an empty filename when dePbo'd.

The last 'entry' should be blank defining the next byte and all bytes thereafter to be the data block. However, resistance format pbo's *sometimes* obscure this.

## Pbo Header Entry

A standard pbo entry as as follows

```
struct entry
{
 Asciiz  filename; //a zero terminated string defining the path and filename,
                   //         relative to the name of this pbo.
                   //Zero length filenames ('\0') indicate first (optional), or last (
                   //  Other fields in the last entry are filled by zero bytes.
  .
 ulong   PackingMethod; //0x00000000 uncompressed
                        //0x43707273 packed
                        //0x56657273 Product Entry (resistance/elite/arma)
 ulong   OriginalSize;  // Unpacked: 0 or same value as the DataSize
                        // Packed: Size of file after unpacking.
                        // This value is needed for byte boundary unpacking
                        // since unpacking itself can lead to bleeding of up
                        // to 7 extra bytes.
 ulong   Reserved;
 ulong   TimeStamp;     // meant to be the unix filetime of Jan 1 1970 +, but often 0
 ulong   DataSize;      // The size in the data block.
                        // This is also the file size when not packed
};
```

### Null Entries

Entries with no file name indicate *boundaries*. The obvious one being end of header.

There are two 'boundaries' used in pbo headers.

1. A header extension, found only in Resistance/arma/elite style pbo's, and
2. End of header

An end of header is (of course) mandatory. It is normally indicated by all other entries also being zero in the struct. However, a sometimes seen case is a 'signature' of 0x43707273 in the compression method for the pbo overall. An indication, that some, none, or all, of the pbo is compressed. Somewhat useless.

The truth of the matter is that it doesn't matter muchly. Detection of the end of header, and, when applied, detection of a start of header, is indicated by no file name. The content of these entries is immaterial, the engine makes no use of them. However, certain 3rd party addon makers rely on the fact that *most* pbo extraction tools expect fields to be zero (even though they don't matter). As such, this prevents _some_ pbo's from being extracted by those tools.

## HeaderExtension

A Header Extension occurs as the first entry on all NON CWC pbo's

If present (and it *is* optional) it is the FIRST entry in the header. It *extends* the entry!

```
struct entry
{
 // standard entry
 Asciiz  filename; // = 0
 ulong   PackingMethod; //=0x56657273 Product Entry (resistance/elite/arma)
 ulong   OriginalSize;  // =0
 ulong   Reserved;//=0
 ulong   TimeStamp;//=0
 ulong   DataSize; //=0
// end of 'standard' entry
 struct HeaderExtension
 {
  Asciiz String;
  ............
  Asciiz String; // '\0' mandatory last (or only) entry
 };
};
```

Note especially that *some* addon suppliers provide non zero fields in either or both of these special entries to confuse DePbo tools. The 'key' that cannot be got round is that a zero length filename means, a special entry.

There can be as many Strings as, well, as many as, a piece of string!!!!

There are as many string entries as the tool that creates the pbo chooses to put in there!

The LAST (or only!) String is a zero length asciiz string. Eg '\0'

Resistance and Arma Pbo's only use three string entries (the last entry being '\0')

However addon makers do attempt to confuse depbo tools by putting more, or less! string entries in this struct to break the tool. Beware, true resistance/arma pbo's utilise 3 entries, but the engine will accept any amount (one or more)

### Resistance Pbo

Resistance Pbo's add an *optional* EntryType as the first entry type in the header.

The meaning of the following entry (the 2nd one in the header) changes to:

```
struct ProductEntry
{
   Asciiz    *EntryName;      // = "product"
   Asciiz    *ProductName;    // = "OFP: Resistance"
   Asciiz    *ProductVersion; // = ""
};
```

This extended entry is a set-in-concrete signature for Resistance pbo's. It is not employed by the engine. But See Arma Elite comments.

### OFP Elite Pbo

Operation Flashpoint: Elite pbo's intended for use on the Xbox are identical in makeup to Resistance pbo's except for the following TWO differences.

1) The 'Resistance' header entry has changed to the following.

```
struct ProductEntry
{
  Asciiz    *EntryName;      // = "prefix"
  Asciiz    *ProductName;    // = "<AddonFileName>"
  Asciiz    *ProductVersion; // = ""
};
```

2) Five *Additional* bytes exist at end of the contiguous data block. This is probably a checksum but has not been verified.

Note that for *Operation Flashpoint Elite* and Armed Assault the compression cannot be used. This is because of the requirement to be able to stream data from the files.

Note that <*AddonFileName*> refers to *the* name of the <file>.pbo. It is moot whether a fully qualified pathname is used (MP mision play), or not (general, DVD based, adddons).

### ArmA Pbo

Armed Assault Pbo's are currently Identical in makeup as Operation Flashpoint Elite Pbo's except for the following difference

The 5 additonal 'checksum' bytes at end of an Elite pbo's contiguous data block have expanded to 21 bytes. This is a file hash. The 1st byte of either of these types of pbo is always a leading zero.

The altered Resistance header, first introduced in Elite, specifies a 'virtual' file reference versus the actual name of the pbo (which could change).

The traditional method in OFP to access external addons from another addon is

model=\AnotherAddon\SomeModel.p3d;

In Arma, this has changed to

model=\AnotherVirtualAddon\SomeModel.p3d;

The practicalities of which are that most 3rd party model makers will 'see' no difference since they wont alter the prefix (virtual) name from that of the pbo itself.

## Data compression

Data compression in ofp is a mild, but effective, form of run length encoding (LZH), allowing (up to) 4k of previous data to repeat itself.

Compression is indicated when a signature of 0x43707273 and the filesizes do not match in the entry.

The following code also applies to the packing method employed in wrp (OPRW) and pac/paa files which have no header info simply a block of known output length that must be decoded. In all cases, the OUTPUT size is known. With pbo's, the INPUT size is only a boundary definition to the next block of compressed data. It is not used or relevant to decoding data because (up to) 7 residual bytes could exist in the last flag word of the block. As such, only the fixed in concrete output size is relevant.

The compressed data block is in contiguous 'packets' of different lengths

```
block {packet1}...{packetN} {4 byte checksum}
.
packet
{
   byte    Format;
   byte    packetdata[...];      // no fixed length
}
```

The contents of the packetdata contain mixtures of raw data that is passed directly to the output, and, 2byte pointers.

Format: bit values determine what the packetdata is. It is interpeted lsb first thus;

```
BitN =1    -          append byte directly to file (read single byte)
BitN= 0    -          pointer (read two bytes)
```

for example:

format byte, is 0x45, binary notation is: 01000101.

There are three bytes in the block a little further past the format flag that will be passed directly to the output when encountered, and there are FIVE pointers.

In this example, first byte of packetdata is passed to output, 2 bytes are read to make a pointer, next byte is passed (ultimately) to output and so on.

```
For the very last packet in the block, it is almost inevitable that there will be
excessive bits. These are ignored (truncated) as the final output length is always
known from the Entry. You cannot rely on the ignored bits in the format flag (up to se
of them) to be any particular value (0 or 1).
```

A pointer consists of a 12 bits address and 4 bit run length.

The pointer is a reference to somewhere in the previous 4k max of built output. Given Intel's endian word format the bytes b1 and b2 form a short word value B2B1

The format of B2B1 is unfortunately AAAA LLLL AAAAAAAA, requiring a bit of shift mask fiddling.

The address refers to the start of some data in the currently rebuilt part of the file. It is a value, relative to the current length of the reconstructed part of the file (FL).

The run length of the data to be copied, the 'pattern' has 4 bits and therefore, in theory, 0 to 15 bytes can be duplicated. In practice the values are 3..18 bytes because copying 0,1 or 2 bytes makes no sense.

Relative position (rpos) into the currently built output is calculated as

```
rpos = FL  - ((B2B1 &0x00FF) + (B2B1 & 0xF000)>>4) )
```

The length of the data block: rlen

```
rlen = (B2B1 & 0x0F00>>8) + 3
```

With the values of rpos and rlen there are three basic situations possible:

```
rpos + rlen < FL // bytes to copy are within the existing reconstructed data
block is added to the end of the file, giving a new length of FL = FL + rlen.
```

rpos + rlen > FL // data to copy exceeds what's available

```
In this situation the data block has a length of FL – rpos and it is added to the rec
```

rpos + rlen < 0 This is a special case where spaces are added to the decoded file until FL = FL,Initial + rlen

---

The checksum, the last four bytes of any compressed data block. It is an unsigned long (Intel Little Endian order). It is simply a byte-at-a-time, unsigned additive spillover of the decompressed data.

Each and every compressed data block, contains it's own, unique checksum.

There is no, checksum, or other protective device, employed on a pbo overall. Exceptions: Elite and Arma have residual data after the end of contiguous data block that do, represent, a signature for the file.

## Bibliography

```
DosTools : http://dev-heaven.net/projects/list_files/mikero-pbodll
cpbo : http://www.kegetys.net/arma/
```



Categories: ArmA: File Formats | Operation Flashpoint Elite: Editing | Operation Flashpoint: Missions | Operation Flashpoint: Editing | BIS File Formats

---


MINIDAYZ
THIS IS YOUR STORY.
THIS IS MINIDAYZ.
LOG IN & PLAY