



Navigation

- [Main Page](#)
- [Source SDK index](#)
- [Recent changes](#)
- [Random page](#)

Support

- [Getting help](#)
- [Source SDK FAQ](#)
- [Level Design FAQ](#)
- [SDK Help Forums](#)

Steam Community

- [Source SDK Hub](#)
- [Steam Games](#)

Tools

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)
- [Page information](#)

Page

Discussion

View

View source

History


VPK File Format

The [VPK](#) file format is a package format used by post-GCF Source engine games to store content.

Contents [\[hide\]](#)

- 1 Conception
- 2 Features
 - 2.1 Preload Data
 - 2.2 Multiple Archives
- 3 Versions
- 4 File Format
 - 4.1 Directory
 - 4.1.1 Header
 - 4.1.1.1 VPK 1
 - 4.1.1.2 VPK 2
 - 4.1.2 Tree
 - 4.1.3 Footer
 - 4.1.4 VPK 2 Sections
 - 4.1.4.1 File data
 - 4.1.4.2 Archive MD5 checksums
 - 4.1.4.3 Other MD5 checksums
 - 4.1.4.4 Public Key & Signature
 - 4.2 Archive

Conception

Prior to [Left 4 Dead](#), typical Source engine games stored their content in [GCF](#) files. Executable files, modifiable files (e.g. configuration files) and custom content were copied and stored locally on the user's hard drive. Possibly brought on by [poor performance](#) , the [NCF](#) file format was introduced and all game content was copied entirely to the hard drive. This, however, introduced a new problem. Source engine materials and models are stored in thousands of small files and it would be expensive to continuously open and close these files. The solution was the conception of the VPK file format which is used to store Left 4 Dead materials, models and particles in a handful of files which can be quickly accessed.

Features

Preload Data

In order to efficiently access small or critical files, the beginning of each file can optionally be stored in the VPK directory. In practice, this seems to be limited to the first 1000 bytes of Source engine materials ([VMT](#) files) which are typically only a few hundred bytes in size.

Multiple Archives

Previous Source engine games that had been distributed by the more advanced GCF file format had the luxury of internally fragmenting new and updated files. This meant that new and updated files could be efficiently downloaded and saved with minimal bandwidth and disk IO. Because the new VPK files are independent of distribution (Left 4 Dead is distributed by NCF files and Steam knows nothing of the VPK file format), their content is split up over multiple archives that seem to be limited to about 32 MB in size.

Because of this, when a file in a specific file is updated, only the VPK archive that contains the file needs to

be updated. Additionally, new files can be downloaded to their own individual archives. This is why most of the newer archives are small in size; their contents are limited the the files added in a single update.

Versions

1

Alien Swarm
Dota 2
Left 4 Dead
Left 4 Dead 2
Portal 2
Portal Stories: Mel
Source Filmmaker

2

Counter-Strike: Global Offensive
Counter-Strike: Source
Day of Defeat: Source
Half-Life: Source
Half-Life 2
Half-Life 2: Deathmatch
Portal
Team Fortress 2

Third-Party

Titanfall

File Format

A VPK package is actually spread out over multiple files sharing the same extension. The directory is stored in a specific file called <name>_dir.vpk and the content is spread over several additional archive files called <name>_*.vpk (where * is the zero based archive index). Consequentially, there are two file formats:

Directory

Header

VPK 1

Originally, the VPK file had no header or identifier. This changed when the [June 25, 2009 Left 4 Dead update](#) was released adding support for third party campaigns. VPK directory files created after this date have the following header:

```
struct VPKHeader_v1
{
    const unsigned int Signature = 0x55aa1234;
    const unsigned int Version = 1;

    // The size, in bytes, of the directory tree
    unsigned int TreeSize;
};
```

If the file data is stored in the same file as the directory, its offset is (sizeof(VPKHeader_v1) + TreeLength).

VPK 2

```
struct VPKHeader_v2
{
    const unsigned int Signature = 0x55aa1234;
    const unsigned int Version = 2;

    // The size, in bytes, of the directory tree
    unsigned int TreeSize;
```

```

// How many bytes of file content are stored in this VPK file (0 i
unsigned int FileDataSectionSize;

// The size, in bytes, of the section containing MD5 checksums for
unsigned int ArchiveMD5SectionSize;

// The size, in bytes, of the section containing MD5 checksums for
unsigned int OtherMD5SectionSize;

// The size, in bytes, of the section containing the public key an
unsigned int SignatureSectionSize;

};

```

If the file data is stored in the same file as the directory, its offset is (sizeof(VPKHeader_v2) + TreeLength).

Tree

The format of the directory tree is a little unorthodox. It consists of a tree three levels deep that seems to be structured for file size. The first level of the tree consists of file extensions (e.g. *vmt*, *vtf* and *mdl*), the second level consists of directory paths (e.g. *materials/brick*, *materials/decals/asphalt* and *models/infected*), and the third level consists of file names, file information and preload data. Each tree node begins with a null terminated ASCII string and empty strings are used to signify the end of a parent node. Pseudo-code to read the directory might look something like:

```

ReadString(file)
    string = ""
    while true
        char = ReadChar(file)
        if char = null
            return string
        string = string + char

```

```

ReadDirectory(file)

```

```

while true
    extension = ReadString(file)
    if extension = ""
        break
    while true
        path = ReadString(file)
        if path = ""
            break
        while true
            filename = ReadString(file)
            if filename = ""
                break
            ReadFileInformationAndPreloadData(file)

```

A nonexistent extension (in example/file), path (in example.txt), or filename is represented by a single space.

Immediately following the null terminator for the filename is the following structure:

```

struct VPKDirectoryEntry
{
    unsigned int CRC; // A 32bit CRC of the file's data.
    unsigned short PreloadBytes; // The number of bytes contained in t

    // A zero based index of the archive this file's data is contained
    // If 0x7fff, the data follows the directory.
    unsigned short ArchiveIndex;

    // If ArchiveIndex is 0x7fff, the offset of the file data relative
    // Otherwise, the offset of the data from the start of the specifi
    unsigned int EntryOffset;

    // If zero, the entire file is stored in the preload data.
    // Otherwise, the number of bytes stored starting at EntryOffset.
    unsigned int EntryLength;

    const unsigned short Terminator = 0xffff;

```

```
};
```

If a file contains preload data, the preload data immediately follows the above structure. The entire size of a file is `PreloadBytes + EntryLength`.

Footer

VPK2 adds a footer section that contains extra CRC data for `pure mode` "so the dedicated servers do not need to compute them at startup but can be checked with the command `sv_pure_checkvpk`".

VPK 2 Sections

File data

This can be read like a separate VPK archive embedded in the directory file (see `EntryOffset` comment in `VPKDirectoryEntry`).

Archive MD5 checksums

This section is an array of these:

```
struct VPK_ArchiveMD5SectionEntry
{
    unsigned int ArchiveIndex;
    unsigned int StartingOffset; // where to start reading bytes
    unsigned int Count; // how many bytes to check
    char MD5Checksum[16]; // expected checksum
}
```

Since `sizeof(VPK_MD5SectionEntry)` is 28, the section size must be a multiple of 28.

Valve's VPK tool refers to these as cache line hashes. They can be checked against the file content with the `checkhash` command.

Other MD5 checksums

```
struct VPK_OtherMD5Section
```

```
{  
    char TreeChecksum[16];  
    char ArchiveMD5SectionChecksum[16];  
    char Unknown[16];  
}
```

Public Key & Signature

```
struct VPK_SignatureSection  
{  
    unsigned int PublicKeySize; // always seen as 160 (0xA0) bytes  
    char PublicKey[PublicKeySize];  
  
    unsigned int SignatureSize; // always seen as 128 (0x80) bytes  
    char Signature[SignatureSize];  
}
```

This section can be viewed with Valve's VPK tool's `dumpsig` command.

Games with this section (296 bytes):

- Counter-Strike Source
- Day of Deafeat: Source
- Half-Life: Source
- Half-Life 2
- Half-Life 2: Deathmatch
- Half-Life 2: Episode 1
- Half-Life 2: Episode 2
- Half-Life 2: Lost Coast
- Portal
- Team Fortress 2

Games lacking this section:

- Counter Strike: Global Offensive

All VPKs tested were from the English Language

Archive

VPK Archives store raw file data. They have no identifying header and know nothing of their contents. Though not necessary, the raw file data is typically tightly packed.

To do: Rewrite and merge Notes section to the format section

Notes




Valve apparently added skipping to the specifications. I found out when trying to write my own VPK parser in C#. This should be merged with the format area later, and reworded.



Valve uses nulls to signify if skipping is used. On a normal entry, it uses 2 nulls, and is followed by the format above. However, there are cases where there are only one, or no nulls at the start, and this means that some level of skipping is used.

If there are 2 nulls, no skipping is used, and the extension, path, and name are read as usual. If there is 1 null, the extension is skipped (It's the same extension as the last read entry), and then the path and name are read as usual. If there are no nulls, the extension and path are the same as the last entry (Skipped), and only the name is read.

This system has only been observed in VPK1.

VPK readers

- Rick's VPK Tool  (v1)
- GCFScape (v1/v2)
- VPK File Format/vpk2 reader.py (v2)
- HLBox17b  (v2)
- jvpk  (v1/v2)

- [Titanfall VPK Tool](#)  (v3)
- [VPK Python module](#)  (v1/v2)

See also

- [L4D Campaign Add-on Tutorial](#) (VPK authoring.)
- [VPK](#)
- [GCFscape](#) (VPK file viewer.)
- [HLLib](#) (Example VPK source code.)

Category: [File formats](#)

This page was last modified on 27 August 2015, at 23:16.

This page has been accessed 131,700 times.

[About Valve Developer Community](#) [Terms of Use](#) [Third Party Legal Notices](#)

