ZIP64 extension support study

Lukáš Hejl (lukas.hejl@prusa3d.cz)
Vojtěch Bubník (vojtech.bubnik@prusa3d.cz)

Introduction

The original ZIP format has three limitations, which are the maximum size of each uncompressed file is 4 GiB (2^32 bytes), the total size of a ZIP file is 4GiB (offsets in **Central directory file header** and in **End of central directory record** are only 32-bits), and the maximum number of files/records is 65,535. The PKZIP specification (https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT) since version 4.5 has introduced the ZIP64 extension, which allows increasing all these limits. The current PKWARE PKZIP specification version is 6.3.9. **Which minimum PKZIP specification should the core 3MF specification refer to? Anyways, we shall require a ZIP64 extension.**

According to the PKZIP specification, the use of the ZIP64 extension should be indicated by the PKZIP specification version 0x2D (4.5) or newer in the **version needed to extract (minimum)** field of both the **Local file header** and the **Central directory file header**. Also, for all file sizes and file offsets, which are overridden by the ZIP64 extension fields, their original 32 bit counterparts have to be filled with 0xFFFFFFFF (32-bit) or 0xFFFF (16-bit).

According to the PKZIP specification, the ZIP64 extension can be used even when none of the limits are exceeded.

Previous work

The report by Materialize from September 2018 pinpoints no ZIP64 support by Microsoft 3D Builder and OpenXML Package editor for Visual Studio, however MS Word OPC library supports ZIP64 just fine.
https://github.com/3MFConsortium/archived_documents/blob/master/MeetingMaterials_44/Zip64_Issues_Materialise.zip

zipdetails tool (https://github.com/pmqs/IO-Compress)

This tool reads the headers of a ZIP file and displays them in a human readable form. We used it to compare ZIP headers. This tool can also perform some validity checks of the ZIP headers. We recommend this tool to learn the details and quirks of the PKZIP specifications. The tool is available in Linux distros, however we recommend to use the latest version from github.

Tested applications

- 7Zip
- Ashampoo (similar to WinRAR or 7Zip)
- Cura (FDM slicer by Ultimaker)
- CraftWare (FDM slicer)
- DotNetZip
- File Roller (Ubuntu Archive Manager)
- ideaMaker (FDM slicer)
- Info-ZIP (default zip command on Ubuntu)
- Microsoft 3D Builder
- PeaZip (general purpose GUI compressor / decompressor, open source)
- PKZIP
- Simplify3D (FDM slicer)
- Windows Explorer
- WinRar
- WinZip
- PrusaSlicer (miniz library extended to support ZIP64 extension and streaming)

Common information

Most of the tested tools used the ZIP64 extension only on demand when one of the limits of the standard ZIP format is exceeded, which helps to ensure better compatibility across tools. The exception is the DotNetZip tool, where the ZIP64 extension can be enforced at the time of saving even for smaller files.

Except for Microsoft 3D Builder, Microsoft STL repair, and CraftWare, all tools are able to extract ZIP files with the ZIP64 extension without any problems, including the ZIP64 streaming extension.

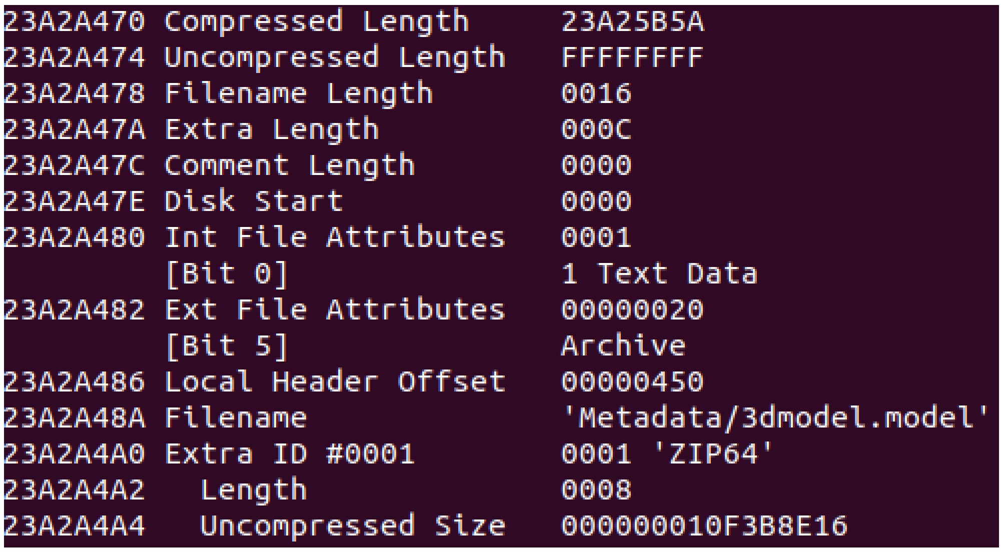
All tools correctly fill in the **version needed to extract (minimum)** field in the **Local file header** and **Central directory file header** with the value 0x2D (version 4.5), which is required for the ZIP64 extension.

Also, all tools fill in the **compressed size** and the **uncompressed size** fields in the **Local file header** with 0xFFFFFFFF when the ZIP64 extension is used, and the actual file sizes and offsets are then provided by the ZIP64 extension **Extra field**.

File Roller, PeaZip, PKZip, 7Zip, WinZip, and Windows Explorer are conservative in usage of the ZIP64 extension Extra fields, which are only used if their corresponding ZIP32 fields would overflow. Thus for example if a file is larger than 4GiB but is compressed below 4GiB, the ZIP64 extension is applied to an uncompressed size only, not to a compressed size. This applies to both the **Local header** and the **End of central directory record.**

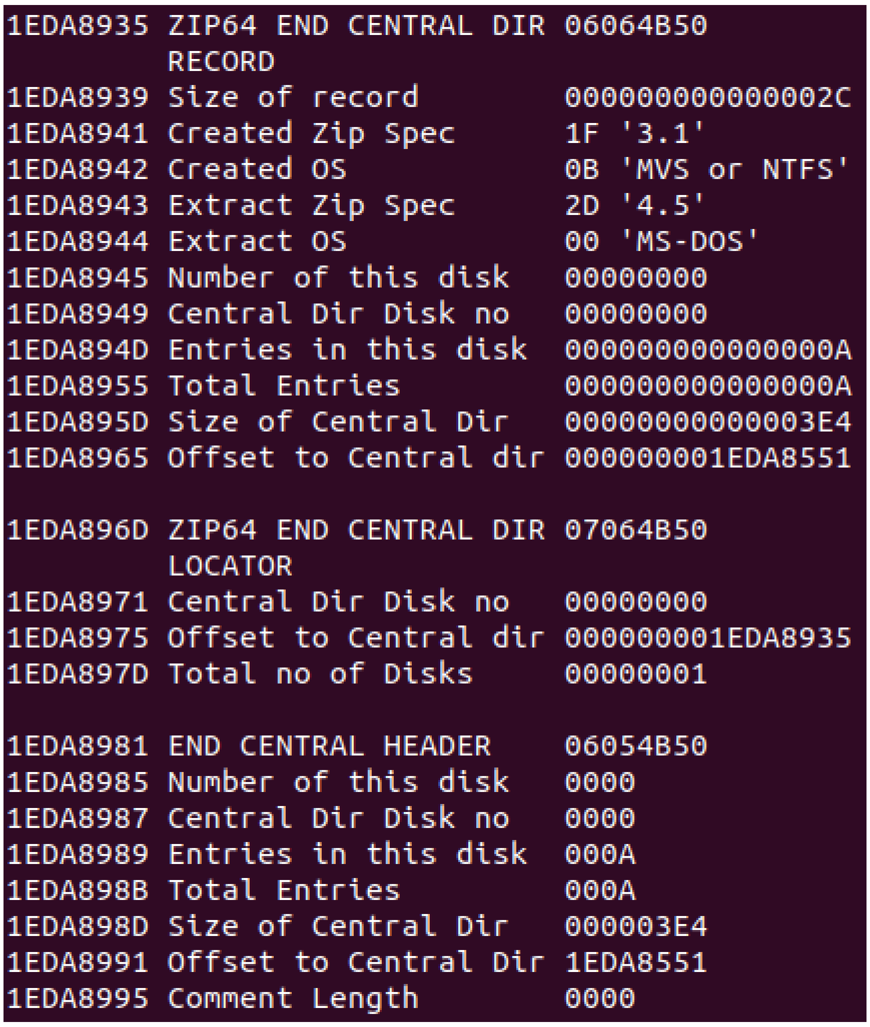
Ashampoo and DotNetZip always store all the ZIP64 extension fields for a ZIP64 archive even if they are not needed.

The following ZIP64 **Central directory file header** shows an example, where the ZIP64 extension is only used for the uncompressed file size, not for a compressed file size.



```
23A2A470 Compressed Length      23A25B5A
23A2A474 Uncompressed Length    FFFFFFFF
23A2A478 Filename Length        0016
23A2A47A Extra Length           000C
23A2A47C Comment Length         0000
23A2A47E Disk Start             0000
23A2A480 Int File Attributes    0001
         [Bit 0]                1 Text Data
23A2A482 Ext File Attributes    00000020
         [Bit 5]                Archive
23A2A486 Local Header Offset    00000450
23A2A48A Filename               'Metadata/3dmodel.model'
23A2A4A0 Extra ID #0001         0001 'ZIP64'
23A2A4A2   Length               0008
23A2A4A4   Uncompressed Size    000000010F3B8E16
```

Info-ZIP and WinRar create a **ZIP64 End of central directory record** even if all values fit into the **End of central directory record**. In this case, they fill in their respective 32 bit fields with real file lengths and offsets, thus ZIP64 **End of central directory record** is redundant, as shown in the following table:


```
1EDA8935 ZIP64 END CENTRAL DIR 06064B50
         RECORD
1EDA8939 Size of record        000000000000002C
1EDA8941 Created Zip Spec      1F '3.1'
1EDA8942 Created OS            0B 'MVS or NTFS'
1EDA8943 Extract Zip Spec      2D '4.5'
1EDA8944 Extract OS            00 'MS-DOS'
1EDA8945 Number of this disk   00000000
1EDA8949 Central Dir Disk no   00000000
1EDA894D Entries in this disk  000000000000000A
1EDA8955 Total Entries         000000000000000A
1EDA895D Size of Central Dir   00000000000003E4
1EDA8965 Offset to Central dir 000000001EDA8551

1EDA896D ZIP64 END CENTRAL DIR 07064B50
         LOCATOR
1EDA8971 Central Dir Disk no   00000000
1EDA8975 Offset to Central dir 000000001EDA8935
1EDA897D Total no of Disks     00000001

1EDA8981 END CENTRAL HEADER    06054B50
1EDA8985 Number of this disk   0000
1EDA8987 Central Dir Disk no   0000
1EDA8989 Entries in this disk  000A
1EDA898B Total Entries         000A
1EDA898D Size of Central Dir   000003E4
1EDA8991 Offset to Central Dir 1EDA8551
1EDA8995 Comment Length        0000
```

Info-ZIP

If zip tool is called with a streaming command similar to **cat file.txt | zip file.zip -,** then because the input file size is not known in advance, the tool chooses a combination of ZIP64 extension and streaming. This means, Info-ZIP always uses a ZIP64 extension in a streaming mode.

File Roller

It uses streaming to store all files and also takes advantage of knowing the size of all files in advance. That means, the ZIP64 extension is only used when needed, streaming extension is always used and the file sizes and offsets are stored both at the start of the Local file header and after the file block using the streaming extension.

7Zip

7Zip considers ZIP64 files produced by DotNetZip corrupted, because DotNetZip emits all possible ZIP64 extension fields into the **Central directory file header**, not just the compressed and uncompressed size like most other compressors. We believe the DotNetZip files conform to the PKZIP ZIP64 specification though.

Windows Explorer

It has a problem with modifying ZIP files (we tested deleting files from a ZIP archive, we expect adding files will show the same issues) if the ZIP was stored with a ZIP64 extension and the ZIP in its compressed form is larger than 4 GiB. If the compressed ZIP is smaller than 4 GiB but contains uncompressed files larger than 4GiB, the Windows Explorer can only modify ZIP files that do not use a **ZIP64 End of central directory record**. Or if they use a **ZIP64 End of central directory record**, then the ZIP64 End of central directory record must match the **End of central directory record**, which is the case of ZIP64 files produced by Info-ZIP (Ubuntu Linux default command line zip tool) and WinRar. Most likely the Microsoft Explorer just ignores the ZIP64 End of central directory record in that case.

Ashampoo

For most ZIP files that use the ZIP64 extension, if you try to remove one of the files inside the ZIP archive with this tool, some other files are accidentally removed. When the ZIP64 extension is not used, this problem does not happen. Plus for Windows Explorer, it at least does not corrupt the archive.

Ashampoo always stores the ZIP64 uncompressed and compressed sizes into the **Central directory file header** even if they are not needed (their 32 bit counterparts are sufficient). In addition, Ashampoo stores a ZIP64 field **Offset of local header record**, which no other tool saves.

DotNetZip

For files smaller than 4GB, it stores an Extra Field entry with ID 0x9999 into the **Local file header**, probably to preallocate space to overwrite this block with a ZIP64 extension block.

When using the ZIP64 extension, DotNetZip stores an unusually long Extra Field into the **Central directory file header**: 28 bytes entry of the payload (32 bytes including size and ID of the block) where it lists the usual uncompressed and compressed file size plus some additional fields that no other compressors store. 7Zip marks this as a corrupt header, although it is OK.

Microsoft 3D Builder, Microsoft STL repair, and CraftWare

Cannot load any 3MF file containing a ZIP64 extension.

Cura, ideaMaker, and Simplify3D

Can open 3MF files with ZIP64 extension without problems.

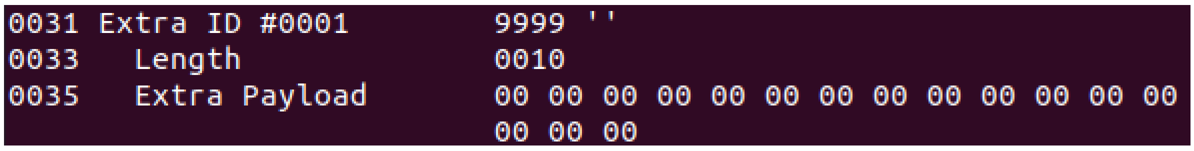
PrusaSlicer (miniz library extended to support ZIP64 extension and streaming)

We can open ZIP files (3MF) from all the mentioned tools without any problems, including ZIP64 extension or streaming extension.

In PrusaSlicer we use streaming because we do not know in advance the size of the model XML file we are generating on line while storing into the ZIP archive. Because of that, we also always used to use the ZIP64 extension. But as we found out, some tools have a problem with the ZIP64 extension.

To improve compatibility between all tools, we modified the miniz library to allow the ZIP64 extension to be used in streaming mode only in case one of the file size limits is exceeded. Our strategy is based on creating the ZIP file without the ZIP64 extension first, but preallocating a custom block with an ID (0x9999) in the Extra field of the Local file header. After finishing writing the compressed file block, if the ZIP64 extension is found to be necessary, we seek back in the ZIP file and rewrite the preallocated custom block with the ZIP64 extension block. In addition, we adjust the minimum required PKZIP version number in the Local file header and we rewrite the 32 bit file sizes and offsets with FFFFs.

The following block shows the phony custom block we stored into the Local file header to be optionally rewritten with a ZIP64 block.



```
0031 Extra ID #0001        9999 ''
0033   Length              0010
0035   Extra Payload       00 00 00 00 00 00 00 00 00 00 00 00 00
                           00 00 00
```

The DotNetZip tool probably uses a similar method to preallocate a phony custom block, so we are using the same ID 0x9999 of the custom block as DotNetZip does. We also verified that none of the ZIP64 consumers complain about this custom block in the Local file header in case it is not being overwritten by the ZIP64 block. Indeed, according to the PKZIP specification, if the tool doesn't understand the ID Extra field, it should skip it.

UTF-8

Introduction

According to the PKZIP specification, the file names should be encoded using IBM Code Page 437 (CP437) by default. CP437 only supports western languages, thus the ZIP specification has been extended to store filenames using UTF-8 encoding.

Two different approaches to store filenames in UTF-8 encoding were introduced:

1) Set the 11th bit in **the general purpose bit flag**, which indicates that UTF-8 encoding will be used instead of CP437 for filenames and comments.
2) Store an UTF-8 encoded file name inside Info-ZIP Unicode Path Extra Field (ID 0x7075). In this case, the 11th bit in **the general purpose bit flag** is not set. This method is backward compatible with old ZIP consumers that do not understand the UTF-8 extension, as an alternate CP437 file name may be stored the old way and a legacy ZIP consumer may ignore the UTF-8 file names stored in extra fields.

Tested applications

- 7Zip
- Ashampoo (similar to WinRAR or 7Zip)
- DotNetZip
- File Roller (Ubuntu Archive Manager)
- Info-ZIP (default zip command on Ubuntu)
- PeaZip (general purpose GUI compressor / decompressor, open source)
- PKZIP
- Windows Explorer
- WinRar
- WinZip

Saving file names with correct encoding

Saving file names with correct encoding turned out to be a major issue for many ZIP producers. However, if the file names were correctly stored using UTF-8 encoding, most of the tested ZIP consumers had no issue extracting the files, including the correctly encoded file names.

Windows Explorer

Windows Explorer does not use UTF-8 for any filename, CP437 encoding is always used. Windows Explorer even fails to convert characters representable in CP437 into CP437. Maybe it just stores the file names using the local 8-bit Windows encoding?

7Zip and PeaZip

Both tools have the same exact problem with the encoding of input filenames. For example, for the name 'ÇüéâäàåçêëèïîìÄÅ', they set the 11th bit in **the general purpose bit flag** and save them correctly in UTF-8. However, they do not set the 11th bit for the name 'áéěíóýščřžňť”, saving the file name as if it was encoded in CP437.

Ashampoo, DotNetZip, File Roller, and Info-ZIP

Ashampoo, DotNetZip (with forced UTF-8 saving), File Roller, and Info-ZIP all set the 11th bit and correctly encode file names in UTF-8.

WinZip

WinZip correctly fills in the Info-ZIP Unicode Path Extra Field (ID 0x7075) for all names instead of setting the 11th bit in **the general purpose bit flag**.

WinRAR

WinRar sets the 11th bit in **the general purpose bit flag** for some UTF-8 encoded filenames and stores the UTF-8 file name in the headers, while for some other UTF-8 encoded file name within the same ZIP file it emits an **Info-ZIP Unicode Path Extra Field** (ID 0x7075) to store UTF-8 encoded filename. The rule by which WinRAR chooses between the two options is unknown to us. None of the ZIP consumers had an issue with that strategy.

PKZIP

PKZIP does not set UTF-8 encoding for any file name, IBM Code Page 437 encoding is always used, and characters that cannot be encoded using IBM Code Page 437 are replaced with question marks.

Extracting files with names in the correct encoding

Only a sample of ZIP archives containing UTF-8 encoded filenames was used for the following test. These ZIP files were produced by Ashampoo, WinRar, WinZip, File Roller, and Info-ZIP. The ZIP files produced by other tools mostly had the zipped file names unrecoverable due to incorrect file name encoding, so the ZIP produced by these other tools were not used for the following tests.

- WinZip, WinRar, Windows Explorer, 7Zip, PeaZip, and File Roller extracts file names correctly in all cases.
- The Ashampoo tool displays the file names incorrectly in GUI, but it extracts all file names correctly.
- The DotNetZip tool fails to extract UTF-8 encoded file names correctly for all tested ZIP files.
- The Info-Zip tool fails to extract UTF-8 encoded file names produced by the Ashampoo tool, but Info-Zip extracts UTF-8 encoded file names from archives produced by other tools just fine. We don’t know why Info-Zip fails on Ashampoo

archives, we don’t see significant differences between Ashampoo generated ZIPs and ZIPs generated by other tools.