# The Format of the Guild Wars 2 Archive File

Jon Dahm

January 3, 2014

## Contents

# Notes

## Libraries

To my knowledge, there are two major C++ libraries for working with the Archive file. Github user Ahom has created a library for working with File Records and extracting images that you can find here. Github user Rhoot has created a library that will extract information from a large number of files within the Archive. You can find his work here. Most of the information in this document has come from these projects.

## Endianness and Numbers

All numbers I list in this document are decimal (base 10) unless specified otherwise. Hexadecimal numbers are followed by a subscript x ($\texttt{1A}_x$). Sometimes a single byte will be listed as a character rather than a number. In these cases the value of that byte is the ASCII code of the character listed.

When I list values, sometimes I will list them as full numbers (like $\texttt{40CB}_x$) and sometimes I will list them as individual bytes (like [$\texttt{CB}_x$,$\texttt{40}_x$]). When I list the individual bytes, they are listed in the order they appear in the Archive. When I list them as full numbers, that is their actual value.

The Archive is arranged in little-endian format. This means that if you see a 16-bit value [$\texttt{CB}_x$,$\texttt{40}_x$], its actual value is $\texttt{40CB}_x$.

## Disclaimer

I do not condone use of this document to modify the archive for any reason. Modifying the archive is a direct violation of the Terms of Service you agreed to follow when you bought the game.

# 1 File Records

This chapter will introduce you to the main portions of the Archive, from which you can find every file represented within. After reading this chapter, you should be able to produce a list of all files within the archive. Additionally, if a file within the archive is referenced by its ID, you should be able to retrieve it.

## 1.1 The Archive Header

The Archive begins with a 40-byte header which describes some of the properties of the Archive and points to the Main File Table. The format of this header can be found in Table 1.

Table 1: the Archive header

| Byte | Size | Value | Description |
|---|---|---|---|
| 0 | 1 | Version | Version of the Archive. Seems to always be $97_x$ |
| 1 | 3 | Identifier | Identifies this file as the Archive file, as opposed to a MS Word file. Always $[45_x, 4E_x, 1A_x]$. |
| 4 | 4 | Header Size | Size of this header. Always 40. |
| 8 | 4 | (unknown) | Always $CABA0001_x$. |
| 12 | 4 | Chunk Size | Size of each chunk in the file. Always 512. |
| 16 | 4 | (unknown)[1] | Always $8ED0A720_x$. |
| 20 | 4 | (unknown) | Always $00040002_x$. |
| 24 | 8 | MFT Offset | The offset from the beginning of the Archive to the Main File Table. |
| 32 | 4 | MFT Size | Size of the Main File Table in bytes. |
| 36 | 4 | (unknown) | Always 0. |

## 1.2 The Main File Table

The Main File Table (MFT) is a list of all of the files in the Archive. Its structure begins with a 24-byte-long header, whose format is given in Table 2. The header is followed by a number of 24-byte entries that make up the

table. Each entry refers to a single file and some associated metadata. The entries are not listed in any particular order. See Table 3 for details.

The first fifteen entries in the MFT are reserved for special files in the Archive. They are documented below:

| | |
|---|---|
| 1 | Archive Header |
| 2 | File ID Table (See Section 1.3) |
| 3 | MFT (self reference) |
| 4–15 | Blank Entries |

Table 2: the MFT header

| Byte | Size | Value | Description |
|---|---|---|---|
| 0 | 4 | Identifier | Identifies the start of the MFT. Always `['M','f','t',1A`$_x$`]`. |
| 4 | 8 | (unknown) | |
| 12 | 4 | Length | Number of entries in the table plus one. |
| 16 | 8 | (unknown) | Always 0. |

Table 3: an MFT entry

| Byte | Size | Value | Description |
|---|---|---|---|
| 0 | 8 | Offset | Offset from the beginning of the Archive to the start of the file. |
| 8 | 4 | Archived Size | Size in bytes of the file within the archive. |
| 12 | 2 | Compression | Type of compression the file is under. See below. |
| 14 | 2 | Flags | Other flags. See below. |
| 16 | 4 | (unknown) | Always 0. |
| 20 | 4 | (unknown) | Always `4867 4BC7`$_x$. |

Valid values for Compression:

| | |
|---|---|
| 0 | Uncompressed |
| 8 | Huffman Compression |

Valid values for Flags:

| | |
|---|---|
| 1 | In Use |
| 2 | (unknown) |

### 1.3 The File ID Table

The File ID Table gives each file in the MFT an ID. Each entry in the table has the format listed in table 4. The entries are not listed in any particular order.

For the most part, each entry has only one ID. However, many have more than one ID each. As of the time of this writing, approximately a third of the files in the Archive have two IDs, and none have more. *More research must be done into why some entries have multiple IDs.*

Additionally, some entries may contain nil values for either field. I haven't found a significant number of these, but they exist. I have only found entries where both fields are nil, and none where only one was nil. My recommendation is to discard any entries with nil fields.

Table 4: a File ID Table entry

| **Byte** | **Size** | **Value** | **Description** |
|---|---|---|---|
| 0 | 4 | File ID | |
| 4 | 4 | MFT Entry Index | Indices start at 1 |

# 2 Files and Compression

This chapter will introduce you to how to identify files and decompress files that have been compressed. Additionally, I'll discuss the compression used on many of the texture files in the Archive. After reading this chapter, you should be able to, given the address of the start of a file, provide its raw data, whether the file was compressed or not.

## 2.1 File Types

Every file starts with an 8-byte header identifying the type of file and how large it is. The first 4 bytes of the header are the file's type identifier, typically represented by four character codes (4CC). The second 4 bytes tell you how long the uncompressed file is, if the file is compressed.

In the latest version of the Archive at the time of this writing, 99% of the files were compressed. All of these files are represented in the general file header by one 4CC. To find the actual 4CC defining the file type, you have to decompress the file, which we will go over in the next section.

The following table describes all 4CCs that appear in the general file header, listed in decreasing order of frequency:

| | |
|---|---|
| [$08_x$,$00_x$,$01_x$,$80_x$] | Compressed File |
| ['A','T','E','X'] | General Use Texture |
| ['A','T','E','U'] | UI Texture |
| ['K','B','2','f'] | (unknown) |
| ['K','B','2','g'] | (unknown) |
| [$7C_x$,$1A_x$,'I','z'] | (unknown) |
| [$97_x$,'A','N',$1A_x$] | (unknown) |