



OpenOffice.org's Documentation of the Microsoft Compound Document File Format

Author	Daniel Rentz ✉ mailto:dr@openoffice.org 🌐 http://sc.openoffice.org
License	Public Documentation License
Contributors	
Other sources	Hyperlinks to Wikipedia (🌐 http://www.wikipedia.org) for various extended information
Mailing list	✉ mailto:dev@sc.openoffice.org Subscription ✉ mailto:dev-subscribe@sc.openoffice.org
Download	PDF 🌐 http://sc.openoffice.org/compdocfileformat.pdf XML 🌐 http://sc.openoffice.org/compdocfileformat.sxw
Project started	2004-Aug-30
Last change	2005-Mar-24
Revision	1.2

Contents

1	Introduction	3
1.1	License Notices	3
1.2	Abstract	3
1.3	Used Terms, Symbols, and Formatting	4
2	Storages and Streams	5
3	Sectors and Sector Chains	6
3.1	Sectors and Sector Identifiers	6
3.2	Sector Chains and SID Chains	7
4	Compound Document Header	8
4.1	Compound Document Header Contents	8
4.2	Byte Order	9
4.3	Sector File Offsets	9
5	Sector Allocation	10
5.1	Master Sector Allocation Table	10
5.2	Sector Allocation Table	11
6	Short-Streams	12
6.1	Short-Stream Container Stream	12
6.2	Short-Sector Allocation Table	12
7	Directory	13
7.1	Directory Structure	13
7.2	Directory Entries	15
8	Example	17
8.1	Compound Document Header	17
8.2	Master Sector Allocation Table	19
8.3	Sector Allocation Table	19
8.4	Short-Sector Allocation Table	20
8.5	Directory	21
9	Glossary	24

1 Introduction

1.1 License Notices

1.1.1 Public Documentation License Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the "License"); you may only use this Documentation if you comply with the terms of this License. A copy of the License is available at <http://www.openoffice.org/licenses/PDL.html>.

The Original Documentation is "OpenOffice.org's Documentation of the Microsoft Compound Document File Format".

The Initial Writer of the Original Documentation is Sun Microsystems, Inc., Copyright © 2003. All Rights Reserved.

See title page for Author contact and Contributors.

All Trademarks are properties of their respective owners.

1.1.2 Wikipedia

Wikipedia Disclaimer: http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer

1.2 Abstract

This document contains a description of the binary format of Microsoft Compound Document files.

Compound document files are used to structure the contents of a document in the file. It is possible to divide the data into several *streams*, and to store these streams in different *storages* in the file. This way compound document files support a complete file system inside the file, the streams are like files in a real file system, and the storages are like sub directories.

1.3 Used Terms, Symbols, and Formatting

• References

A reference to another chapter is symbolised by a little arrow: →1.1.

• Examples

An example is indented and marked with a light-grey border.

This is an example.

• Numbers and Strings

Numerical values are shown in several number systems:

Number system	Marking	Example
Decimal	None	1234
Hexadecimal	Trailing “H”	1234 _H
Binary	Trailing “2”	1001 ₂

Constant strings are enclosed in quotation marks. They may contain specific values (control characters, unprintable characters). These values are enclosed in angle brackets.

Example of a string containing a control character: “abcdef<01_H>ghij”.

• Content Listings

- The term “*Not used*” means: Ignore the data on import and write zero bytes on export. The same applies for unmentioned bits in bit fields.
- The term “*Unknown*” describes data fields with fixed but unknown contents. On export these fields have to be written as shown.
- At several places a variable is introduced, which represents the value of this field for later use, e.g. in formulas. An example can be found in →4.1.

• Formulas

Important formulas are shown in a light-grey box.

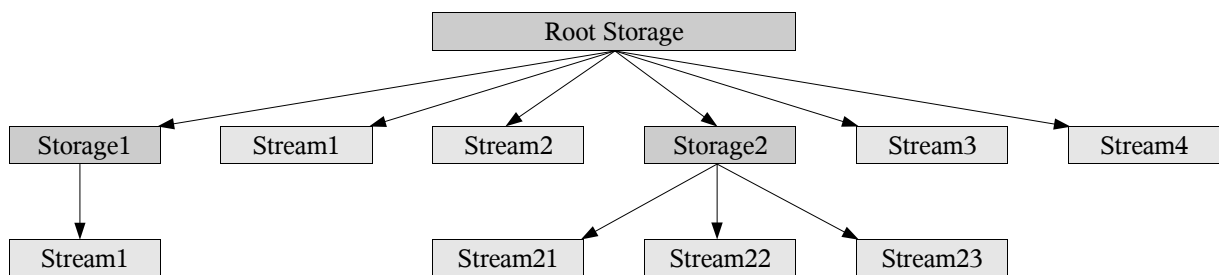
2 Storages and Streams

Compound document files work similar to real file systems. They contain a number of independent data *streams* (like files in a file system) which are organised in a hierarchy of *storages* (like sub directories in a file system).

Storages and streams are named. The names of all storages and streams that are direct members of a storage must be different. Names of streams or storages that are members of different storages may be equal.

Each compound document file contains a *root storage* that is the direct or indirect parent of all other storages and streams.

Example of a storage/stream hierarchy. The names of all direct members of a storage must be different, but it is possible that two different storages contain a stream named “Stream1”.



3 Sectors and Sector Chains

3.1 Sectors and Sector Identifiers

All streams of a compound document file are divided into small blocks of data, called *sectors*. Sectors may contain internal control data of the compound document or parts of the user data.

The entire file consists of a header structure (the compound document header, →4.1) and a list of all sectors following the header. The size of the sectors can be set in the header and is fixed for all sectors then.

HEADER
SECTOR 0
SECTOR 1
SECTOR 2
SECTOR 3
SECTOR 4
SECTOR 5
SECTOR 6
⋮

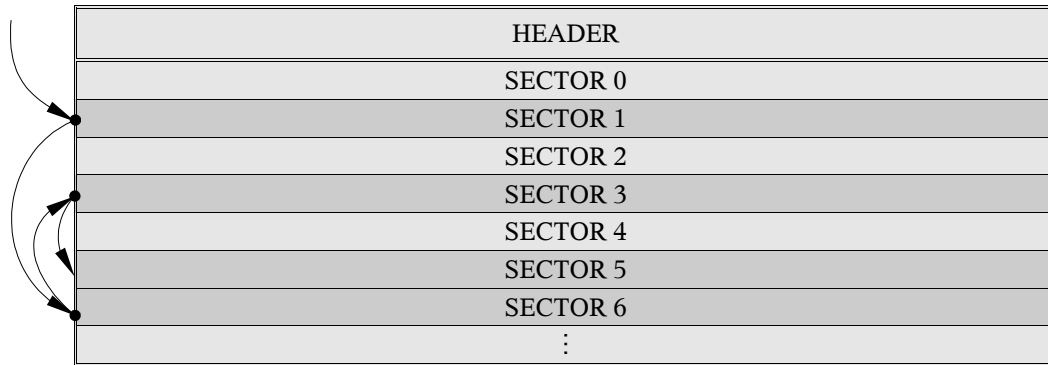
Sectors are enumerated simply by their order in the file. The (zero-based) index of a sector is called *sector identifier* (SID). SIDs are *signed 32-bit integer values*. If a SID is not negative, it must refer to an existing sector. If a SID is negative, it has a special meaning. The following table shows all valid special SIDs:

SID	Name	Meaning
−1	<i>Free SID</i>	Free sector, may exist in the file, but is not part of any stream
−2	<i>End Of Chain SID</i>	Leading SID in a SID chain (→3.2)
−3	<i>SAT SID</i>	Sector is used by the sector allocation table (→5.2)
−4	<i>MSAT SID</i>	Sector is used by the master sector allocation table (→5.1)

3.2 Sector Chains and SID Chains

The list of all sectors used to store the data of one stream is called *sector chain*. The sectors may appear unordered and may be located on different positions in the file. Therefore an array of SIDs, the *SID chain*, specifies the order of all sectors of a stream. A SID chain is always terminated by a special *End Of Chain SID* with the value -2 (→3.1).

Example: A stream consists of 4 sectors. The SID chain of the stream is $[1, 6, 3, 5, -2]$. See →4.3 on how to calculate the file offset of a sector from its SID.



The SID chain for each stream is built up from the sector allocation table (→5.2), with exception of short-streams (→6) and the following two internal streams:

- the master sector allocation table (→5.1), which builds its SID chain from itself (each sector contains the SID of the following sector), and
- the sector allocation table itself, which builds its SID chain from the master sector allocation table.

4 Compound Document Header

The *compound document header* (simply “header” in the following) contains all data needed to start reading a compound document file.

4.1 Compound Document Header Contents

The header is always located at the beginning of the file, and its size is exactly 512 bytes. This implies that the first sector (with SID 0) always starts at file offset 512.

Contents of the compound document header structure:

Offset	Size	Contents
0	8	Compound document file identifier: D0 _H CF _H 11 _H E0 _H A1 _H B1 _H 1A _H E1 _H
8	16	Unique identifier (UID) of this file (not of interest in the following, may be all 0)
24	2	Revision number of the file format (most used is 003E _H)
26	2	Version number of the file format (most used is 0003 _H)
28	2	Byte order identifier (→4.2): FE _H FF _H = Little-Endian FF _H FE _H = Big-Endian
30	2	Size of a sector in the compound document file (→3.1) in power-of-two (<u>ssz</u>), real sector size is <u>sec_size</u> = 2 ^{<u>ssz</u>} bytes (most used value is 9 which means 512 bytes, minimum value is 7 which means 128 bytes)
32	2	Size of a short-sector in the short-stream container stream (→6.1) in power-of-two (<u>sssz</u>), real short-sector size is <u>short_sec_size</u> = 2 ^{<u>sssz</u>} bytes (most used value is 6 which means 64 bytes, maximum value is sector size <u>ssz</u> , see above)
34	10	Not used
44	4	Total number of sectors used for the sector allocation table (→5.2)
48	4	SID of first sector of the directory stream (→7)
52	4	Not used
56	4	Minimum size of a standard stream (in bytes, most used size is 4096 bytes), streams smaller than this value are stored as short-streams (→6)
60	4	SID of first sector of the short-sector allocation table (→6.2), or -2 (<i>End Of Chain SID</i> , →3.1) if not extant
64	4	Total number of sectors used for the short-sector allocation table (→6.2)
68	4	SID of first sector of the master sector allocation table (→5.1), or -2 (<i>End Of Chain SID</i> , →3.1) if no additional sectors used
72	4	Total number of sectors used for the master sector allocation table (→5.1)
76	436	First part of the master sector allocation table (→5.1) containing 109 SIDs

4.2 Byte Order

All data items containing more than one byte may be stored using the Little-Endian or Big-Endian method¹, but in real world applications only the Little-Endian method is used. The Little-Endian method stores the least significant byte first and the most significant byte last. This applies for all data types like 16-bit integers, 32-bit integers, and Unicode characters.

Example: The 32-bit integer value $13579BDF_H$ is converted into the Little-Endian byte sequence $DF_H\ 9B_H\ 57_H\ 13_H$, or to the Big-Endian byte sequence $13_H\ 57_H\ 9B_H\ DF_H$.

4.3 Sector File Offsets

With the values from the header it is possible to calculate a file offset from a SID:

$$\text{sec_pos}(\text{SID}) = 512 + \text{SID} \cdot \text{sec_size} = 512 + \text{SID} \cdot 2^{\text{ssz}}$$

Example with $\text{ssz} = 10$ and $\text{SID} = 5$:

$$\text{sec_pos}(\text{SID}) = 512 + \text{SID} \cdot 2^{\text{ssz}} = 512 + 5 \cdot 2^{10} = 512 + 5 \cdot 1024 = 5632.$$

¹ For more information see <http://en.wikipedia.org/wiki/Endianness>.

5 Sector Allocation

5.1 Master Sector Allocation Table

The *master sector allocation table* (MSAT) is an array of SIDs of all sectors used by the sector allocation table (SAT, →5.2), which finally is needed to read any other stream in the file. The size of the MSAT (number of SIDs) is equal to the number of sectors used by the SAT. This value is stored in the header (→4.1).

The first 109 SIDs of the MSAT are contained in the header too. If the MSAT contains more than 109 SIDs, additional sectors are used to store the following SIDs. The header contains the SID of the first sector used for the MSAT then (otherwise there is the special *End Of Chain SID* with the value -2 , →3.1).

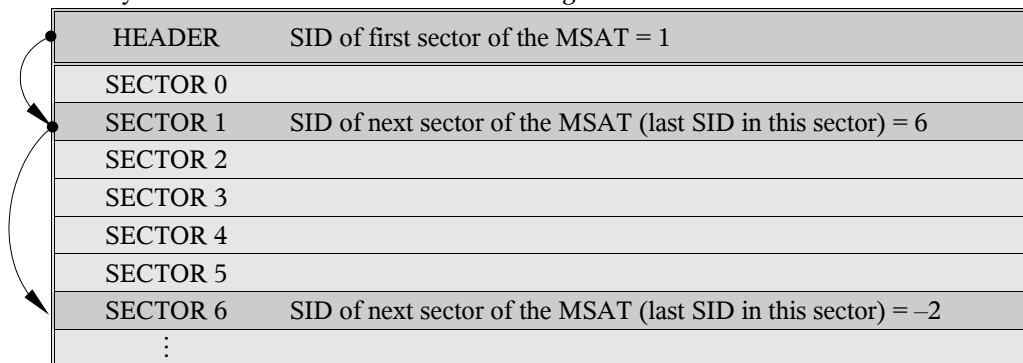
The last SID in each sector of the MSAT refers to the next sector used by the MSAT. If no more sectors follow, the last SID is the special *End Of Chain SID* with the value -2 (→3.1).

Contents of a sector of the MSAT (sec_size is the size of a sector in bytes, see →4.1):

Offset	Size	Contents
0	<u>sec_size</u> - 4	Array of (<u>sec_size</u> - 4) / 4 SIDs of the MSAT
<u>sec_size</u> - 4	4	SID of the next sector used for the MSAT, or -2 if this is the last sector

The last sector of the MSAT may not be used completely. Unused space is filled with the special *Free SID* with the value -1 (→3.1). The MSAT is build up by concatenating all SIDs from the header and the additional MSAT sectors.

Example: A compound document file contains a SAT that needs 300 sectors to be stored. The header specifies a sector size of 512 bytes. This implies that a sector is able to store 128 SIDs. The MSAT consists of 300 SIDs (number of sectors used for the SAT). The first 109 SIDs are stored in the header. The remaining 191 SIDs of the MSAT need additional two sectors. In this example the first sector of the MSAT may be sector 1 which contains the next 127 SIDs of the MSAT (the 128th SID points to the next MSAT sector), and the second sector of the MSAT may be sector 6 which contains the remaining 64 SIDs.



5.2 Sector Allocation Table

The *sector allocation table* (SAT) is an array of SIDs. It contains the SID chain (→3.2) of all user streams (except short-streams, →6) and of the remaining internal control streams (the short-stream container stream, →6.1, the short-sector allocation table, →6.2, and the directory, →7). The size of the SAT (number of SIDs) is equal to the number of existing sectors in the compound document file.

5.2.1 Reading the Sector Allocation Table

The SAT is built by reading and concatenating the contents of all sectors given in the MSAT (→5.1). The sectors have to be read according to the order of the SIDs in the MSAT.

Contents of a sector of the SAT (*sec_size* is the size of a sector in bytes, see →4.1):

Offset	Size	Contents
0	<i>sec_size</i>	Array of <i>sec_size</i> /4 SIDs of the SAT

5.2.2 Using the Sector Allocation Table

When building a SID chain (→3.2) for a specific stream, the *current position* (array index) in the SAT array refers to the current sector, while the *SID contained at this position* specifies the following sector in the sector chain.

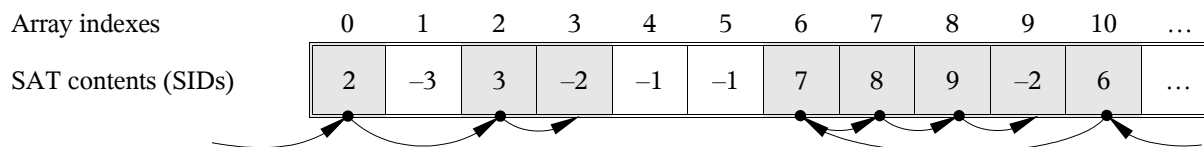
The SAT may contain special *Free SIDs* with the value -1 (→3.1) at any position. These sectors are not used by a stream. The position referring to the last sector of a stream contains the special *End Of Chain SID* with the value -2 . Sectors used by the SAT itself are not chained, but are marked with the special *SAT SID* with the value -3 . Finally, sectors used by the MSAT are marked with the special *MSAT SID* with the value -4 .

The entry point of a SID chain has to be obtained somewhere else, e.g. from the directory entry (→7.2) of a user stream, or from the header (→4.1) for internal control streams such as the short-sector allocation table (→6.2), or the directory stream itself (→7.1).

Example: A compound document file contains one sector needed for the SAT (sector 1) and two streams. Sector 1 contains the SID array of the SAT shown below. The SAT contains the special *SAT SID* (value -3) at position 1 which marks this sector being part of the SAT.

One stream is the internal directory stream. In this example, the header may specify that it starts with sector 0. The SAT contains the SID 2 at position 0, the SID 3 at position 2, and the SID -2 at position 3. Therefore the SID chain of the directory stream is $[0, 2, 3, -2]$, and the directory stream is stored in 3 sectors.

The directory contains (amongst others) the entry of a user stream that may start with sector 10. This results in the SID chain $[10, 6, 7, 8, 9, -2]$ for this stream.



6 Short-Streams

Whenever a stream is shorter than a specific length (specified in the header, →4.1), it is stored as a *short-stream*. Short-streams do not directly use sectors to store their data, but are all embedded in a specific internal control stream, the short-stream container stream.

6.1 Short-Stream Container Stream

The *short-stream container stream* is stored like any other (long) user stream: The first used sector has to be obtained from the root storage entry in the directory (→7.2), and its SID chain (→3.2) is contained in the SAT (→5.2). The data of all sectors used by the short-stream container stream are concatenated in order of its SID chain. In the next step this stream is virtually divided into short-sectors, similar to sectors in the main compound document file (→3.1), but without a header structure. Therefore the first short-sector (with SID 0) is always located at offset 0 inside the short-stream container stream. The size of the short-sectors is contained in the header (→4.1). With this information it is possible to calculate an offset in the short-stream container stream from a SID:

$$\text{short_sec_pos}(\text{SID}) = \text{SID} \cdot \text{short_sec_size} = \text{SID} \cdot 2^{\text{sssz}}$$

Example with $\text{sssz} = 6$ and $\text{SID} = 5$:

$$\text{short_sec_pos}(\text{SID}) = \text{SID} \cdot 2^{\text{sssz}} = 5 \cdot 2^6 = 5 \cdot 64 = 320.$$

6.2 Short-Sector Allocation Table

The *short-sector allocation table* (SSAT) is an array of SIDs and contains the SID chains (→3.2) of all short-streams, similar to the sector allocation table (→5.2) that contains the SID chains of standard streams.

The first SID of the SSAT is contained in the header (→4.1), the remaining SID chain is contained in the SAT. The SSAT is built by reading and concatenating the contents of all sectors.

Contents of a sector of the SSAT (sec_size is the size of a sector in bytes, see →4.1):

Offset	Size	Contents
0	sec_size	Array of $\text{sec_size}/4$ SIDs of the SSAT

The SSAT will be used similarly to the SAT (→5.2) with the difference that the SID chains refer to short-sectors in the short-stream container stream (→6.1).

7 Directory

7.1 Directory Structure

The *directory* is an internal control stream that consists of an array of *directory entries* (→7.2). Each directory entry refers to a storage or a stream in the compound document file (→2). Directory entries are enumerated in order of their appearance in the stream. The zero-based index of a directory entry is called *directory entry identifier* (DID).

DIRECTORY ENTRY 0
DIRECTORY ENTRY 1
DIRECTORY ENTRY 2
DIRECTORY ENTRY 3
⋮

The position of a directory entry will not change as long as the referred storage or stream exists in the compound document. This implies that the DID of a storage or stream never changes regardless how many other objects are inserted to or removed from the compound document. If a storage or stream is removed, the corresponding directory entry is marked as empty. There is a special directory entry at the beginning of the directory (with the DID 0). It represents the root storage and is called *root storage entry*.

The directory organises direct members (storages and streams) of each storage in a separate red-black tree². Shortly, nodes in a red-black tree have to fulfil *all* of the following conditions:

- The root node is black.
- The parent of a red node is black.
- The paths from the root node to all leaves contain the same number of black nodes.
- The left child of a node is less than the node, the right child is greater.

But note that not all implementations follow these rules. The safest way to read directory entries is to ignore the node colours and to rebuild the red-black tree from scratch.

Example: Taking the example from →2, the directory would have the following structure:

- The root storage is represented by the root storage entry. It does not have a parent directory entry, therefore there are no other entries that can be organised in a red-black tree.
- All members of the root storage (“Storage1”, “Storage2”, “Stream1”, “Stream2”, “Stream3”, and “Stream4”) are inserted into a red-black tree. The DID of the root node of this tree is stored in the root storage entry.
- The storage “Storage1” contains one member “Stream1” which is inserted into a separate red-black tree. The directory entry of “Storage1” contains the DID of “Stream1”.
- The storage “Storage2” contains three members “Stream21”, “Stream22”, and “Stream23”. These directory entries are organised in a separate red-black tree. The directory entry of “Storage2” contains the DID of the root node of this tree.

² See http://en.wikipedia.org/wiki/Red_black_tree.

This results in the fact that each directory entry contains up to three DIDs: The first is the DID of the left child of the red-black tree containing this entry, the second is the DID of the right child in the tree, and (if this entry is a storage) the third is the DID of the root node of another red-black tree containing all sub streams and sub storages.

Nodes are compared by their names to decide whether they become the left or right child of another node, but in a special way:

- A node is less than another node, if the name is shorter; and greater, if the name is longer.
- If both names have the same length, they are compared character by character (case sensitive).

Examples: The name “VWXYZ” is less than the name “ABCDEFGH” because the length of the former name is shorter (regardless of the fact that the character V is greater than the character A). The name “ABCDE” is less than the name “ABCDFG” because the lengths of both names are equal, and comparing the names shows that the fourth character of the former name is less than the fourth character of the latter name.

7.2 Directory Entries

7.2.1 Directory Entry Structure

The size of each directory entry is exactly 128 bytes. The formula to calculate an offset in the directory stream from a DID is as follows:

$$\text{dir_entry_pos}(\text{DID}) = \text{DID} \cdot 128$$

Contents of the directory entry structure:

Offset	Size	Contents
0	64	Character array of the name of the entry, always 16-bit Unicode characters, with trailing zero character (results in a maximum name length of 31 characters)
64	2	Size of the used area of the character buffer of the name (not character count), including the trailing zero character (e.g. 12 for a name with 5 characters: (5+1)·2 = 12)
66	1	Type of the entry: 00 _H = Empty 01 _H = User storage 02 _H = User stream 03 _H = LockBytes (unknown) 04 _H = Property (unknown) 05 _H = Root storage
67	1	Node colour of the entry: 00 _H = Red 01 _H = Black
68	4	DID of the left child node inside the red-black tree of all direct members of the parent storage (if this entry is a user storage or stream, →7.1), -1 if there is no left child
72	4	DID of the right child node inside the red-black tree of all direct members of the parent storage (if this entry is a user storage or stream, →7.1), -1 if there is no right child
76	4	DID of the root node entry of the red-black tree of all storage members (if this entry is a storage, →7.1), -1 otherwise
80	16	Unique identifier, if this is a storage (not of interest in the following, may be all 0)
96	4	User flags (not of interest in the following, may be all 0)
100	8	Time stamp of creation of this entry (→7.2.3). Most implementations do not write a valid time stamp, but fill up this space with zero bytes.
108	8	Time stamp of last modification of this entry (→7.2.3). Most implementations do not write a valid time stamp, but fill up this space with zero bytes.
116	4	SID of first sector or short-sector, if this entry refers to a stream (→7.2.2), SID of first sector of the short-stream container stream (→6.1), if this is the root storage entry, 0 otherwise
120	4	Total stream size in bytes, if this entry refers to a stream (→7.2.2), total size of the short-stream container stream (→6.1), if this is the root storage entry, 0 otherwise
124	4	Not used

7.2.2 Starting Position of a Stream

The directory entry of a stream contains the SID of the first sector or short-sector containing the stream data. All streams that are shorter than a specific size given in the header (→4.1) are stored as a short-stream, thus inserted into the short-stream container stream (→6.1). In this case the SID specifies the first short-sector inside the short-stream container stream, and the short-sector allocation table (→6.2) is used to build up the SID chain (→3.2) of the stream.

7.2.3 Time Stamp

The *time stamp* field is an unsigned 64-bit integer value that contains the time elapsed since 1601-Jan-01 00:00:00 (Gregorian calendar³). One unit of this value is equal to 100 nanoseconds (10^{-7} seconds). That means, each second the time stamp value will be increased by 10 million units.

When calculating the date from a time stamp, the correct rules of leap year handling have to be respected⁴:

- a year divisible by 4 is a leap year;
- with the exception that a year divisible by 100 is not a leap year (e.g. 1900 was no leap year);
- with the exception that a year divisible by 400 is a leap year (e.g. 2000 was a leap year).

Example: The time stamp value is 01A5E403C2D59C00_H.

Calculation step	Formula	Result
Conversion to decimal		$t_0 = 118,751,670,000,000,000$
Fractional amount of a second	$r_{\text{frac}} = t_0 \text{ modulo } 10,000,000$	$r_{\text{frac}} = 0$
Remaining entire seconds	$t_1 = t_0 / 10,000,000$	$t_1 = 11,875,167,000$
Seconds in a minute	$r_{\text{sec}} = t_1 \text{ modulo } 60$	$r_{\text{sec}} = 0$
Remaining entire minutes	$t_2 = t_1 / 60$	$t_2 = 197,919,450$
Minutes in an hour	$r_{\text{min}} = t_2 \text{ modulo } 60$	$r_{\text{min}} = 30$
Remaining entire hours	$t_3 = t_2 / 60$	$t_3 = 3,298,657$
Hours in a day	$r_{\text{hour}} = t_3 \text{ modulo } 24$	$r_{\text{hour}} = 1$
Remaining entire days	$t_4 = t_3 / 24$	$t_4 = 137,444$
Entire years from 1601-Jan-01 ⁵	$r_{\text{year}} = 1601 + \text{number of full years in } t_4$	$r_{\text{year}} = 1601 + 376 = 1977$
Remaining days in year 1977	$t_5 = t_4 - (\text{number of days from } 1601\text{-Jan-01 to } 1977\text{-Jan-01})$	$t_5 = 137,444 - 137,331 = 113$
Entire months from 1977-Jan-01	$r_{\text{month}} = 1 + \text{number of full months in } t_5$	$r_{\text{month}} = 1 + 3 = 4 = \text{April}$
Remaining days in month April	$t_6 = t_5 - (\text{number of days from } 1977\text{-Jan-01 to } 1977\text{-Apr-01})$	$t_6 = 113 - 90 = 23$
Resulting day of month April	$r_{\text{day}} = 1 + t_6$	$r_{\text{day}} = 1 + 23 = 24$

The final result is 1977-Apr-24 01:30:00. Guess what it is...

³ See http://en.wikipedia.org/wiki/Gregorian_calendar.

⁴ See http://en.wikipedia.org/wiki/Leap_year for some background information.

⁵ You may use your favourite date/time manipulation library to perform the following steps.

8 Example

This chapter shows a possible way to open a compound document file. The file that is processed here is a simple spreadsheet document in Microsoft Excel file format, written by OpenOffice.org Calc.

8.1 Compound Document Header

The first step is to read the compound document header (→4.1). The first 512 bytes of the file may look like this:

```

00000000_H D0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00
00000010_H 00 00 00 00 00 00 00 00 3B 00 03 00 FE FF 09 00
00000020_H 06 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
00000030_H 0A 00 00 00 00 00 00 00 00 10 00 00 02 00 00 00
00000040_H 01 00 00 00 FE FF FF FF 00 00 00 00 00 00 00
00000050_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000060_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000080_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000090_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000A0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000B0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000C0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000D0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000E0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000F0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000100_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000110_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000120_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000130_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000140_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000150_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000160_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000170_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000180_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000190_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000001A0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000001B0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000001C0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000001D0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000001E0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000001F0_H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

```

8 Example

- 1) 8 bytes containing the fixed compound document file identifier:

```
00000000H D0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00
```

- 2) 16 bytes containing a unique identifier, followed by 4 bytes containing a revision number and a version number. These values can be skipped:

```
00000000H D0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00
00000010H 00 00 00 00 00 00 00 00 3B 00 03 00 FE FF 09 00
```

- 3) 2 bytes containing the byte order identifier. It should always consist of the byte sequence FE_H FF_H:

```
00000010H 00 00 00 00 00 00 00 00 00 00 3B 00 03 00 FE FF 09 00
```

- 4) 2 bytes containing the size of sectors, 2 bytes containing the size of short-sectors. The sector size is 512 bytes, and the short-sector size is 64 bytes here:

```
00000010H 00 00 00 00 00 00 00 00 3B 00 03 00 FE FF 09 00
00000020H 06 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
```

- 5) 10 bytes without valid data, can be ignored:

```
00000020H 06 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
```

- 6) 4 bytes containing the number of sectors used by the sector allocation table (→5.2). The SAT uses only one sector here:

```
00000020H 06 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
```

- 7) 4 bytes containing the SID of the first sector used by the directory (→7). The directory starts at sector 10 here:

```
00000030H 0A 00 00 00 00 00 00 00 00 10 00 00 02 00 00 00
```

- 8) 4 bytes without valid data, can be ignored:

```
00000030H 0A 00 00 00 00 00 00 00 00 10 00 00 02 00 00 00
```

- 9) 4 bytes containing the minimum size of standard streams. This size is 00001000_H = 4096 bytes here:

```
00000030H 0A 00 00 00 00 00 00 00 00 10 00 00 02 00 00 00
```

- 10) 4 bytes containing the SID of the first sector of the short-sector allocation table (→6.2), followed by 4 bytes containing the number of sectors used by the SSAT. In this example the SSAT starts at sector 2 and uses one sector:

```
00000030H 0A 00 00 00 00 00 00 00 00 10 00 00 02 00 00 00
00000040H 01 00 00 00 FE FF FF FF 00 00 00 00 00 00 00 00
```

- 11) 4 bytes containing the SID of the first sector of the master sector allocation table (→5.1), followed by 4 bytes containing the number of sectors used by the MSAT. The SID here is -2 (*End Of Chain SID*, →3.1) which states that there is no extended MSAT in this file:

```
00000040H 01 00 00 00 FE FF FF FF 00 00 00 00 00 00 00 00
```

- 12) 436 bytes containing the first 109 SIDs of the MSAT. Only the first SID is valid, because the SAT uses only one sector (see above). Therefore all remaining SIDs are set to the special *Free SID* with the value -1 (→3.1). The only sector used by the SAT is sector 0:

```
00000040H 01 00 00 00 FE FF FF FF 00 00 00 00 00 00 00 00
00000050H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000060H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
⋮ ⋮
```

8.2 Master Sector Allocation Table

The header contains the entire MSAT (→5.1), therefore nothing else can be done here. The MSAT consists of the SID chain [0, -2].

8.3 Sector Allocation Table

To build the SAT (→5.2), all sectors specified in the MSAT have to be read. In this example this is only sector 0. It starts at file offset 00000200_H = 512 (→4.3) and may look like this:

```

00000200H FD FF FF FF FF FF FF FF FE FF FF FF 04 00 00 00
00000210H 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00
00000220H 09 00 00 00 FE FF FF FF 0B 00 00 00 FE FF FF FF
00000230H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000240H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000250H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000260H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000270H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000280H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000290H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000002A0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000002B0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000002C0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000002D0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000002E0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000002F0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000300H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000310H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000320H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000330H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000340H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000350H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000360H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000370H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000380H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000390H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000003A0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000003B0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000003C0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000003D0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000003E0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000003F0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

```

This results in the following SID array for the SAT:

Array indexes	0	1	2	3	4	5	6	7	8	9	10	11	12	...
SID array	-3	-1	-2	4	5	6	7	8	9	-2	11	-2	-1	...

As expected, sector 0 is marked with the special *SAT SID* (→3.1). Sector 1 and all sectors starting with sector 12 are not used (special *Free SID* with value -1).

8.4 Short-Sector Allocation Table

The SSAT (→6.2) starts at sector 2 and consists only of this one sector as specified in the header. This is in line with the SAT that contains the *End Of Chain SID* at position 2. The SID chain of the SSAT is therefore [2, -2]. Sector 2 starts at file offset 00000600_H = 1536 (→4.3) and may look like this:

```

00000600H 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00
00000610H 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00
00000620H 09 00 00 00 0A 00 00 00 0B 00 00 00 0C 00 00 00
00000630H 0D 00 00 00 0E 00 00 00 0F 00 00 00 10 00 00 00
00000640H 11 00 00 00 12 00 00 00 13 00 00 00 14 00 00 00
00000650H 15 00 00 00 16 00 00 00 17 00 00 00 18 00 00 00
00000660H 19 00 00 00 1A 00 00 00 1B 00 00 00 1C 00 00 00
00000670H 1D 00 00 00 1E 00 00 00 1F 00 00 00 20 00 00 00
00000680H 21 00 00 00 22 00 00 00 23 00 00 00 24 00 00 00
00000690H 25 00 00 00 26 00 00 00 27 00 00 00 28 00 00 00
000006A0H 29 00 00 00 2A 00 00 00 2B 00 00 00 2C 00 00 00
000006B0H 2D 00 00 00 FE FF FF FF 2F 00 00 00 FE FF FF FF
000006C0H FE FF FF FF 32 00 00 00 33 00 00 00 34 00 00 00
000006D0H 35 00 00 00 FE FF FF FF FF FF FF FF FF FF FF FF
000006E0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000006F0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000700H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000710H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000720H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000730H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000740H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000750H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000760H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000770H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000780H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000790H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000007A0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000007B0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000007C0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000007D0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000007E0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000007F0H FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

```

This results in the following SID array for the SSAT:

Array indexes	0	1	2	3	4	5	6	7	8	9	10	11	...	41	42	43	44	45	46	47	48	49	50	51	52	53	54	...
SID array	1	2	3	4	5	6	7	8	9	10	11	12	...	42	43	44	45	-2	47	-2	-2	50	51	52	53	-2	-1	...

All short-sectors starting with sector 54 are not used (special *Free SID* with value -1).

8.5 Directory

The header contains the SID of the first sector that contains the directory (→7), which is sector 10 in this example. The directory is always contained in standard sectors, never in short-sectors. Therefore the SAT is used to build up the SID chain of the directory: [10, 11, -2]. Sector 10 starts at file offset $00001600_H = 5632$, and sector 11 starts at file offset $00001800_H = 6144$ (→4.3). In this example, a sector has a size of 512 bytes, therefore each sector contains 4 directory entries (each entry uses exactly 128 bytes), and the entire directory contains 8 entries.

The following example shows what is contained in the first two directory entries (→7.2). The other entries are read accordingly.

8.5.1 Root Storage Entry

The first directory entry always represents the root storage entry. It may look like this:

```
00001600H 52 00 6F 00 6F 00 74 00 20 00 45 00 6E 00 74 00
00001610H 72 00 79 00 00 00 00 00 00 00 00 00 00 00 00 00
00001620H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001630H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001640H 16 00 05 00 FF FF FF FF FF FF FF FF 01 00 00 00
00001650H 10 08 02 00 00 00 00 00 C0 00 00 00 00 00 00 46
00001660H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001670H 00 00 00 00 03 00 00 00 80 0D 00 00 00 00 00 00
```

- 1) 64 bytes containing the character array of the entry name (16-bit characters, terminated by the first <00_H> character). The name of this entry is “Root Entry” here:

```
00001600H 52 00 6F 00 6F 00 74 00 20 00 45 00 6E 00 74 00
00001610H 72 00 79 00 00 00 00 00 00 00 00 00 00 00 00 00
00001620H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001630H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- 2) 2 bytes containing the valid range of the previous character array (22 bytes here, resulting in 10 valid characters):

```
00001640H 16 00 05 00 FF FF FF FF FF FF FF FF 01 00 00 00
```

- 3) 1 byte containing the type of the entry. Must be 05_H for the root storage entry.

```
00001640H 16 00 05 00 FF FF FF FF FF FF FF FF 01 00 00 00
```

- 4) 1 byte containing the node colour of the entry. It is red in this example, breaking the rule that the root storage entry should always be black:

```
00001640H 16 00 05 00 FF FF FF FF FF FF FF FF 01 00 00 00
```

- 5) 4 bytes containing the DID of the left child node, followed by 4 bytes containing the DID of the right child node. Should both be -1 in the root storage entry:

```
00001640H 16 00 05 00 FF FF FF FF FF FF FF FF 01 00 00 00
```

- 6) 4 bytes containing the DID of the root node entry of the red-black tree of all members of the root storage. It is 1 in this example:

```
00001640H 16 00 05 00 FF FF FF FF FF FF FF FF 01 00 00 00
```

- 7) 16 bytes containing a unique identifier, followed by 4 bytes containing additional flags, and two time stamps, 8 bytes each, containing the creation time and last modification time of the storage (→7.2.3). This data can be skipped:

```
00001650H 10 08 02 00 00 00 00 00 C0 00 00 00 00 00 00 46
00001660H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001670H 00 00 00 00 03 00 00 00 80 0D 00 00 00 00 00 00
```

8 Example

- 8) 4 bytes containing the SID of the first sector or short-sector of a stream, followed by 4 bytes containing the stream size. In case of the root storage entry, this is the SID of the first sector and the size of the short-stream container stream (→6.1). It starts at sector 3 and has a size of 0000D80_H = 3456 bytes in this example:

```
00001670H 00 00 00 00 03 00 00 00 80 0D 00 00 00 00 00 00
```

- 9) 4 bytes without valid data, can be skipped:

```
00001670H 00 00 00 00 03 00 00 00 80 0D 00 00 00 00 00 00
```

8.5.2 Second Directory Entry

The second directory entry (with DID 1) may look like this:

```
00001680H 57 00 6F 00 72 00 6B 00 62 00 6F 00 6F 00 6B 00
00001690H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000016A0H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000016B0H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000016C0H 12 00 02 00 02 00 00 00 04 00 00 00 FF FF FF FF
000016D0H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000016E0H 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000016F0H 00 00 00 00 00 00 00 00 51 0B 00 00 00 00 00 00
```

Important data is highlighted. The name of this entry is “Workbook”, it represents a stream, the DID of the left child node is 2, the DID of the right child node is 4, the SID of the first sector is 0, and the stream size is 0000B51_H = 2897 bytes. The stream is shorter than 4096 bytes, therefore it is stored in the short-stream container stream.

8.5.3 Remaining Directory Entries

The remaining directory entries are read similar to the examples above, resulting in the following directory:

DID	Name	Type	DID of left child	DID of right child	DID of first member	SID of first sector	Stream size	Allocation table
0	Root Entry	root	none	none	1	3	3456	SAT
1	Workbook	stream	2	4	none	0	2897	SSAT
2	<01 _H >CompObj	stream	3	none	none	46	73	SSAT
3	<01 _H >Ole	stream	none	none	none	48	20	SSAT
4	<05 _H >SummaryInformation	stream	none	none	none	49	312	SSAT
5		empty						
6		empty						
7		empty						

Starting with the first member directory entry specified in the root storage entry (here: DID 1) it is possible to find all members of the root storage. Entry 1 has two child nodes with DID 2 and DID 4. DID 2 has one child node with DID 3. The child nodes with the DIDs 3 and 4 do not specify more child nodes. Therefore the directory entries with the DIDs 1, 2, 3, and 4 represent members of the root storage.

8.5.4 SID Chains of the Streams

The short-stream container stream (→6.1) is always stored in standard sectors. All user streams are shorter than 4096 bytes (the minimum size of standard streams specified in the header, →4.1), therefore they are stored in the short-stream container stream, and the SSAT is used to build the SID chains of the streams.

DID	Stream name	Allocation table	SID chain
0	<i>Short-stream container stream</i> ⁶	SAT	[3, 4, 5, 6, 7, 8, 9, -2]
1	Workbook	SSAT	[0, 1, 2, 3, 4, 5, ..., 43, 44, 45, -2]
2	<01 _H >CompObj	SSAT	[46, 47, -2]
3	<01 _H >Ole	SSAT	[48, -2]
4	<05 _H >SummaryInformation	SSAT	[49, 50, 51, 52, 53, -2]

8.5.5 Short-Stream Container Stream

The short-stream container stream is read by concatenating all sectors specified in the SID chain of the root storage entry in the directory. In this example, the sectors 3, 4, 5, 6, 7, 8, and 9 have to be read in this order, resulting in a stream with a size of 3584 bytes, but only the first 3456 bytes are used (as specified in the root storage entry). These 3456 bytes are divided into short-sectors with a size of 64 bytes each, resulting in 54 short-sectors.

8.5.6 Reading a Stream

Now the stream “<01_H>CompObj” may be read. The SID chain of this stream is [46, 47, -2], the stream is a short-stream. The two short-sectors 46 and 47 contain the user data. Short-sector 46 starts at offset 2944 in the short-stream container stream, short-sector 47 starts at offset 3008 (→6.1).

⁶ The actual name of this directory entry may be “Root Entry” or similar, but it *refers* to the short-stream container stream too.

9 Glossary

Term	Description	Chapter
Byte order	The order in which single bytes of a bigger data type are represented or stored.	→4.2
Compound document	File format used to store several objects in a single file, objects can be organised hierarchically in <i>storages</i> and <i>streams</i> .	→1.2
Compound document header	Structure in a <i>compound document</i> containing initial settings.	→4.1
Control stream	<i>Stream</i> in a <i>compound document</i> containing internal control data.	→5, →6, →7
DID	Zero-based index of a <i>directory entry</i> (short for “ <i>directory entry identifier</i> ”).	→7.1
Directory	List of <i>directory entries</i> for all <i>storages</i> and <i>streams</i> in a compound document.	→7.1
Directory entry	Part of the directory containing relevant data for a <i>storage</i> or a <i>stream</i> .	→7.2
Directory entry identifier (DID)	Zero-based index of a <i>directory entry</i> .	→7.1
Directory stream	Sector chain containing the directory	→7.1
End Of Chain SID	Special <i>sector identifier</i> used to indicate the end of a <i>SID chain</i> .	→3.1
File offset	Physical position in a file.	→4.3
Free SID	Special <i>sector identifier</i> for unused <i>sectors</i> .	→3.1
Header	Short for “ <i>compound document header</i> ”.	→4.1
Master sector allocation table (MSAT)	<i>SID chain</i> containing <i>sector identifiers</i> of all <i>sectors</i> used by the <i>sector allocation table</i> .	→5.1
MSAT	Short for “ <i>master sector allocation table</i> ”.	→5.1
MSAT SID	Special <i>sector identifier</i> used to indicate that a <i>sector</i> is part of the <i>master sector allocation table</i> .	→3.1
Red-black tree	Tree structure used to organise direct members of a <i>storage</i> .	→7.1
Root storage	Built-in <i>storage</i> that contains all other objects (<i>storages</i> and <i>streams</i>) in a compound document.	→2
Root storage entry	<i>Directory entry</i> representing the <i>root storage</i> .	→7.1
SAT	Short for “ <i>sector allocation table</i> ”.	→5.2
SAT SID	Special <i>sector identifier</i> used to indicate that a <i>sector</i> is part of the <i>sector allocation table</i> .	→3.1
Sector	Part of a compound document with fixed size that contains any kind of <i>stream</i> (user stream or <i>control stream</i>) data.	→3.1
Sector allocation table (SAT)	Array of <i>sector identifiers</i> containing the <i>SID chains</i> of all user <i>streams</i> and a few internal <i>control streams</i> .	→5.2
Sector chain	An array of <i>sectors</i> that forms a <i>stream</i> as a whole.	→3.2
Sector identifier (SID)	Zero-based index of a <i>sector</i> .	→3.1
Short-sector	Part of the <i>short-stream container stream</i> with fixed size that contains one part of a <i>short-stream</i> .	→6.1

Term	Description	Chapter
Short-sector allocation table (SSAT)	Array of <i>sector identifiers</i> containing the <i>SID chains</i> of all <i>short-streams</i> .	→6.2
Short-stream	A user <i>stream</i> shorter than a specific size.	→6
Short-stream container stream	An internal <i>stream</i> that contains all <i>short-streams</i> .	→6.1
SID	Zero-based index of a <i>sector</i> (short for “ <i>sector identifier</i> ”).	→3.1
SID chain	An array of <i>sector identifiers</i> (SIDs) specifying the <i>sectors</i> that are part of a <i>sector chain</i> and thus enumerates all <i>sectors</i> used by a <i>stream</i> .	→3.2
SSAT	Short for “ <i>short-sector allocation table</i> ”.	→6.2
Storage	Part of a <i>compound document</i> used to separate <i>streams</i> into different groups, similar to directories in a file system.	→2
Stream	Part of a <i>compound document</i> containing user data or internal control data, similar to files in a file system.	→2
Stream offset	Virtual position in a <i>stream</i> .	→6.1, →7.2
Time stamp	Value specifying date and time.	→7.2.3
User stream	<i>Stream</i> in a <i>compound document</i> containing user data.	→2